6th SC@RUG 2009 proceedings

Smedinga, Rein; Isenberg, Tobias

SC@RUG 2009 proceedings

Rein Smedinga
Tobias Isenberg
editors

2009
Groningen

ISBN 978-90-367-3867-5
Publisher: Bibliotheek der R.U.
Title: Proceedings 6th Student Colloquium 2008-2009
Computing Science, University of Groningen
NUR-code: 980

Contents

About SC@RUG

Introduction SC@RUG (or student colloquium in full) is a course that master students in computing science follow in the first year of their master study at the University of Groningen.

In the academic year 2008-2009 SC@RUG was organized for the sixth time as a conference. Students wrote a paper, participated in the review process, gave a presentation and were session chair during the conference.

The organizers Tobias Isenberg and Rein Smedinga would like to thank all colleagues, who cooperated in this SC@RUG by collecting sets of papers to be used by the students and by being an expert reviewer during the review process. They also would like to thank Femke Kramer from the Faculty of Arts for her help in organizing this course and Janneke Geertsema for her workshops on presentation techniques and speech therapy.

In these proceedings all accepted papers are published.

Organizational matters SC@RUG 2009 was organized as follows. Students were expected to work in teams of two. The student teams could choose between different sets of papers, that were made available through *Nestor*, the digital learning environment of the university. Each set of papers consisted of about three papers about the same subject (within Computing Science). Some sets of papers contained conflicting opinions. Students were instructed to write a survey paper about this subject including the different approaches in the given papers. The paper should compare the theory in each of the papers in the set and include own conclusions about the subject.
Two teams proposed their own subject.

After submission their papers, each student was assigned one paper to review using a standard review form. The staff member who had provided the set of papers was also asked to fill in such a form. Thus, each paper was reviewed three times (twice by peer reviewers and once by the expert reviewer). Each review form was made available to the authors of the paper through *Nestor*.

All papers could be rewritten and resubmitted, independent of the conclusions from the review. After resubmission each reviewer was asked to re-review the same paper and to conclude whether the paper had improved. Rereviewers could accept or reject a paper. All accepted papers can be found in these proceedings.

All students were asked to present their paper at the conference and act as a chair and discussion leader during one of the other presentations. Half of the participants were asked to organize the of the conference day (i.e., to make the time tables, invite people etc.) The audience graded both the presentation and the chairing and leading the discussion.

Femke Kramer of the Faculty of Arts gave an introductory lecture about what is a scientific conference, and about general aspects of presentation techniques to help the students with their presentation. She also taught a workshop on writing a scientific paper, and one on reviewing scientific papers. Janneke Geertsema gave workshops on presentation techniques and speech therapy that was very well appreciated by the participants.

Students were graded both on all three aspects: the writing process, the review process and the presentation. Writing and rewriting counted for 50% (here we used the grades given by the reviewers and the re-reviewers), the review process itself for 15% and the presentation for 35% (including 5% for the grading of being a chair or discussion leader during the conference). For the grading of the presentations we used the judgements from the audience and calculated the average of these.

In this edition of SC@RUG students were videotaped during their presentation. We used a new tool, made available by the UOCG (Universitair Onderwijscentrum Groningen) to both record the speakers talks and their Powerpoint.

On January 29th, the actual conference took place. Each paper was presented by both authors. That day, we had nine presentations, each consisting of a total of 20 minutes for the presentation and 10 minutes for discussion. As mentioned before, each presenter also had to act as a chair and discussion leader for another presentation during that day. The audience was asked to fill in a questionnaire and grade the presentations, the chairing and leading the discussion. Participants not selected as chair were asked to organize the day.

The head of UOCG, Louwarnoud van der Duim was keynote speaker.

All submitted papers were accepted for this proceedings.

Thanks We could not have achieved the ambitious goal of this course without the invaluable help of the following expert reviewers: Frank Brokken (CIT), Wim Hesselink, Tobias Isenberg, Gerl Moritz, Liang Peng, Gerard Renardel, Jos Roerdink, and Marco Wiering.

Also, the organizers would like to thank the *School of Computing and Cognition* for making it possible to publish these proceedings and the UOCG for sponsoring the conference.

Rein Smedinga
Tobias Isenberg

Comparing tools for static code analysis

Bertjan Broeksema (s1554697), a.h.j.broeksema@student.rug.nl
Frank Hoving (s1651455), f.hoving@student.rug.nl
January 23, 2009

Abstract—Software applications are exposed to growing risks by security flaws. Compiling software with common compilers does not provide the security some programs need. Static analysis is a step beyond compiling, it is a technique to discover security flaws without running the program. There are many tools for static analysis, but what do these tools do and what is their outcome? We give a brief overview of three methods described in the literature, (SPLint[2], Saturn[4] and Eau Claire[1]). Additionally we evaluate a recent tool for applying static analysis named Source Code Analyzer from Fortify[6]. The tool is used to analyze three different software projects. We describe the performance, how issues found are presented to the user and identify some problems with static analysis.

1 Introduction

Insecure software is an increasing risk in a society that heavily depends on large and complex software systems. Software should be reliable and the number of crashes and security breaches must therefore be reduced to a bare minimum. Techniques as cryptography, security protocols and software testing systems[3] are used to make software more secure and reliable. Figure 1 gives an overview of the vulnerabilities in software from 1 January 2001 through 18 September 2001 listed on the Mitre's common vulnerabilities and exposures list[5].

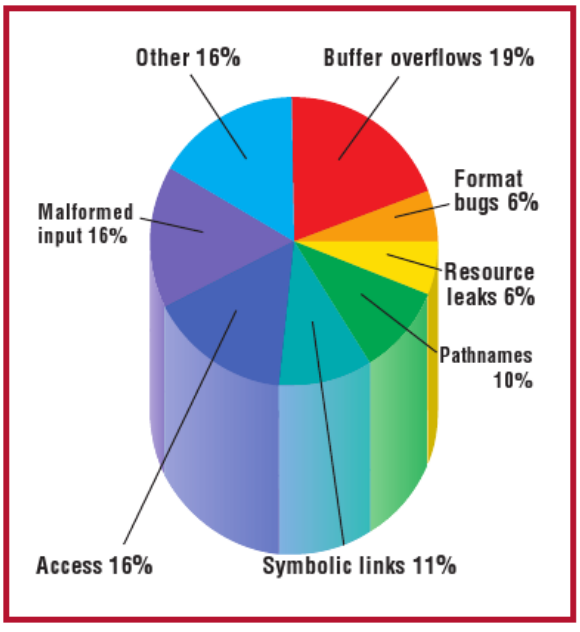


Figure 1: Common vulnerabilities and exposures list for the first nine months of 2001. [2]

In this paper the focus is on static analysis of software. Static analysis is a technique for discovering errors in source code without running the program. In the literature there are several methods [3] for static analysis proposed. Those methods vary from basic text scanning to full code analysis and can be applied programming languages like: C, C++ and JAVA. The simplest methods start with textual scanning of software to find bugs where '=' is used in stead of '=='. However, this simple analysis suffers from many inaccuracies and only discovers basic flaws in the software. The next level of methods contain style checking methods that check variable and function definitions. Even more advanced methods apply semantic-analysis and use syntax trees to check the correctness of the code. Deep flow static analysis is an extension to semantic-analysis and analyzes data-flow and control flow.

To narrow the scope of this paper we constrain the methods to those that apply to the programming language C. This language is still a commonly used language for writing software. The compared methods diverge from lightweight analysis to full scanning analysis. The first method is SPLint [2], which is a variant of Lint and makes use of annotations. Saturn[4] is a boolean satisfiability framework and Eau Claire[1] is a method that uses a theorem prover for finding flaws.

Besides giving an overview of the above mentioned methods we also do an evaluation of a recent program for performing analysis. The evaluation will be done with for the Source Code Analyzer from fortify[6] which is developed by the same author as Eau Claire.

The rest of the paper is structured as follows. In section 2 we describe the introduced tools in more detail. Next, in section 3 we describe the goal and setup of the evaluation. Section 4 describes the results of the evaluation and in section 5 we discuss the results and give some directions for future research.

2 LITERATURE REVIEW

In this section we will look into the tools SPLint, Saturn and Eau Claire with some more detail. For each method we give a summary of the functionality and the theory behind the methods.

2.1 SPLint

According to David Evans and David Larochelle [2], security flaws can be approached in two ways: damage limitation and flaw elimination. Damage limitation is implemented by inserting runtime checks and restrictions to environments where the software runs to limit damage risks. The elimination can be applied by human code review, by testing and by static analysis. SPLint is developed during a research and improves security by extensible lightweight static analysis. This adds some more security after compiling but is less effective and reasonable faster than full analysis. The tool uses annotations in the software associated with function parameters, return values, global variables and structure fields. The annotations describe expected behaviour which is compared to the behaviour of the code itself.

In static analysis there is a trade-off between precision and scalability. The aim of SPLint lies on fast scanning and scalability. By implementing fast scanning and scalability the tool looses precision by skipping security flaws and generating false security flaw warnings during the analysis. The code is screened for buffer overflows which occur often by use of standard string manipulation functions like 'gets' and 'strcpy' which are known to be unsafe.

SPLint provides extensible checking. This is additional checking for user defined vulnerabilities. This makes the tool able to check for user specific or application specific properties. For example, extensible checking can be used by defining taintedness attributes. This attribute is used in [2] to define format bug checking and was discovered in June 2000. Format bugs can be like the construction of the printf function which can give options of writing arbitrary data to the memory when the printf is used wrong. The tool can also use extra annotations assumptions in case library code is unreachable. For example when software uses standard compiler libraries which are not available during the static analysis.

2.2 Saturn

Yichen Xie and Alex Aiken[4] describe another tool for static analysis, named Saturn. This is a boolean satisfiability based framework used for static bug detection. The software is tested on user specified boolean descriptions of functions provided with the code. The framework tests the user expected function description to the code. This framework is constructed from four components. The first component generates a low-level Intermediate Language which models program constructs and saves this in an abstract syntax tree. The second components is a frontend that translates the source code into a call graph in topological order. This graph is written in the intermediate language. The third component transforms expressions and statements into boolean formulas. A property checker uses these formulas to analyze each function and stores the results in a summary database. Finally the discovered violations are compiled into bug reports and the summary database is exported as document of inferred behaviour. This behaviour is mentioned per function.

2.3 Eau Claire

Brian V. Chess[1] describes the concepts behind the tool Eau Claire. This tool uses specifications that describe security properties and handles many common security flaw sensitive construction of the C language like pointers, function calls and arrays. Eau Claire translates the source code into series of verification conditions and presents the verification conditions to a theorem prover. The source code is first translated in Guarded Commands by using the vulnerability specifications. Gaurded Commands form a programming language to describe non-trivial algorithms for formal analyzing. These commands are translated into verification conditions. The conditions will be processed by an automatic theorem prover which will give output to the developer about verification of the functions.

3 EVALUATION SETUP

The aim of this evaluation is to see how a recent application for perfoming static analysis performs and how it presents the results to the developer. At the time of writing Eau Clair is no longer actively maintained or even supported. SPLint is actively supported but needs annotations in the source files to check the software. Saturn needs boolean models for the issues that one is looking for, which are not available at forehand. Because the amount of effort needed for all three of these tools to get reasonable results we decided not to evaluate these tools. Brian Chess provided us with a full version of the Source Code Analyzer (SCA) from Fortify [6].

In this evaluation we use SCA to analyze three software projects of different sizes. It makes use of the concepts introduced before for finding security flaws and additional concepts not presented in this paper. We have looked at different aspects of the tool. The first aspect is the integration of the tool with the build process of the software. The second aspect is the performance in terms of the time it takes to analyze a project with respect to size of the project. The last aspect is the presentation of the different kinds of security issues that are found by the tool.

To compare these aspects three open source software projects developed in C/C++ of different sizes where chosen. Sloccount [7] is used to determine the number of lines for each project. Each project is build using the build system that comes with the project. After the build the code is analyzed using SCA. The unix command time [8] is used to measure the time taken in these two steps to get insight into the relation between the project size and the time it takes to analyze the project and into the relation between build time and analyze time. After the analyze step the results are manually inspected using the graphical tool auditworkbench which is also part of SCA. All steps are performed on a Pentium dual core machine 2.33 GHz with 2 GB of RAM.

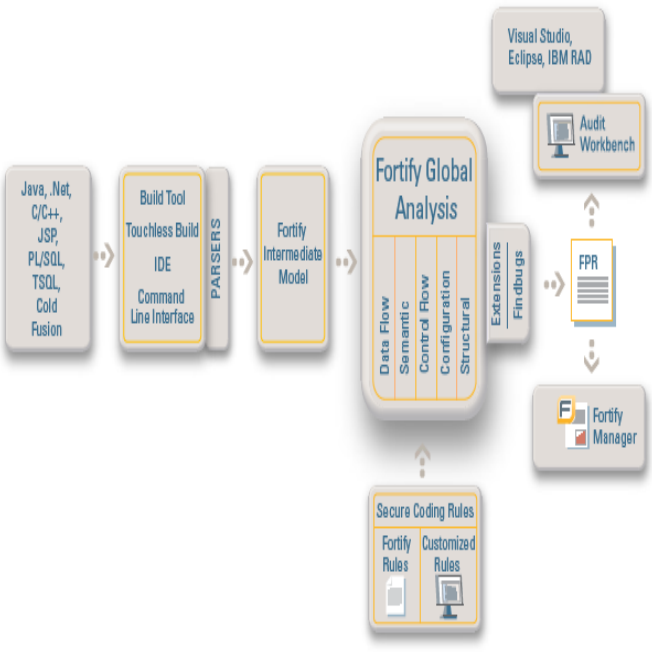


Figure 2: Structure of Fortify SCA

Before presenting the results we first give an overview of SCA. Figure 2 shows the main components of SCA. The Build component shown on the left side of the picture is responsible for making the translation of source code to an intermediate language which is used for further analysis. The build tool is able to connect with the make [9] build system but can also analyze single files. The result of this translation step, call the `Intermediate Fortify Model`, is passed to the `Global Analysis Module`. This module then performs several kinds of analyses like data flow analysis and semantic analysis to find security issues. The kind of issues that the tool can find are described in Secure Coding rules which are also passed to the `Global analysis Module`. By default SCA comes with a huge set of predefined rules. It is also possible to add custom rules to find issues which are specific for the product that is analyzed. The results of these analyses are stored in a special file which can be opened with the auditworkbench tool.

4 EVALUATION OF SCA

We evaluated the source code analyzer by integration and performance which we will discuss to end with the reported issues during the tests.

4.1 Integration with the build system

In order for static analysis tools to be useful they must integrate seamless into the development environment. Preferably the tool integrates with the build system used by the project. This way the tool can use the build system to determine which files must be analyzed and where it can find needed resources (e.g. header files). Some tools, like Splint, do not integrate with the build system but need the mentioned settings as parameters to the splint executable. When a C/C++ project uses the make build system, SCA uses that to determine which files it needs to analyze. This way analyzing project is a matter of just two steps. The first step builds the project and the second step performs the analysis and stores the results in a file:

```
# sourceanalyzer -b buildname make

# sourceanalyzer -b buildname -scan -f results.fpr
```

The results of the analysis are stored in results.fpr which can be opened with the auditworkbench tool. The auditworkbench shows all the issue found. The issues are grouped by severity and within each group ordered by the type of the issue. Issues are linked to the source files where they are found. Each issue is clearly documented and contains references to additional information about the specific kind of issue that was found. Auditworkbench is eclipse based and is also available as an eclipse plug-in. This makes it easy to integrate security analysis into the development environment.

4.2 Performance

To integrate static analysis tools into the development process in a useful manner, it is important that the performance of the tool is known. The time it takes to generate a vulnerability report determines how often the tool can be run. This can range from each build to only before an important release, depending both on the size of the project and the performance of the tool. Table 1 shows the results obtained for each project. The number of lines only include lines of language supported by SCA, which are in C, C++ and java for these projects.

Project	Lines of code	Compile time	Analyze time
GCC	*2.803.087*	*30m40.716s*	*N/A*
Subversion	*605.391*	*8m1.143s*	*50m5.363s*
XMLRPC-C	*45.563*	*1m12.663s*	*5m0.691s*

Table 1: Results of analysis of several projects.

From the results we can conclude that for small and medium projects security analysis can be done on a regular base. It still takes a couple of times the compile time to finish the analysis but on a modern machine projects like XMLRP-C take only a couple of minutes to compile. It is possible to run the analyzer on a single file but in that case the advantage of the integration with the build system is lost. Currently SCA seems not to be able to update the results from a previous run with the modified files.

For GCC we where unable to generate results. After more than six hours the analyzer reported that it had finished 81% of the analysis but did not proceed for more then an hour.

4.3 Reported Issues

SCA uses so called rule packs which contains security rules for supported languages. With the rule packs provided by default a broad range of security issues can be checked for. In this section we will describe some of the issues which where reported on the chosen project.

The most prominent issue reported are buffer overflows. A buffer overflow occurs when data is put into a buffer which is too small. In that case the program may crash or if exploited carefully return addresses might be overwritten which enables an attacker to execute malicious code.

Another kind of issue that is reported does reflect invalid assumptions of the developer. For example functions that operate on **char*** parameters might require that the character sequence ends with the **'\0'** character. Calling these functions with data that comes from user input or from files that might be modified by attackers, are therefore unsafe. The assumption that the developer makes in this case is that the data passed to the function is valid.

Other typical C/C++ issues reported by SCA are about memory management. SCA is able to report dereference of pointer variables that might be null or deleted. It is also able to report error messages when a pointer is freed more than once. Furthermore it reports possible sources of memory leaks.

5 Discussion

In this paper we gave an overview of three methods for static analysis to enhance the security of software. The first method introduced was SPLint which uses annotations to analyse the security of software. We also introduced Saturn, a method which uses a boolean satisfiability framework to detect security flaws. The third method introduced was Eau Clair which uses a theorem prover. We also evaluated the Source Code Analyser from Fortify. We have looked at the performance of SCA for three different open source projects and how the issues which where found are presented to the user.

Although we think that static analysis is an valuable addition to the existing quality and security processes in software development lifecycles, if there are any, we still see some problems. One of these problems problem we clearly identified is the performance issue. For projects with less than 100.000 lines of codes it is quite feasible to run the analyzer for the whole project on a regular base. However when the projects become larger it takes much longer to perform the analysis or eventually the analysis does not complete at all.

Another problem is the initial setup, as we have experienced with SPLint and Saturn. Those tools require a deep understanding of the source code as well as knowledge from the build system and also knowledge from the security issues that one is looking for. Although some of these knowledge may be available this is definitely not always the case. Software development teams might know the source and the build system but may lack knowledge of security issues. On the other hand an external security audit team has deep understanding of the security risks but lack knowledge about the build system or source code of a specific project.

The above mentioned problem relates to another problem, which is the educational effect of the tools used for finding security flaws. Again, tools like SPLint and Saturn can only be used when it is known on forehand what kind of issues needs to be found. So extensive study on these issues, their impact and how to find them using the above mentioned tools is needed. We think that this is a high threshold for introducing static analysis into a development process. SCA takes a very different approach to solve this problem.

By default it supports a wide range of programming languages and provides a large set of rules for finding security issues. Besides this SCA has a comprehensive user interface for doing security audits. In this interface issues are grouped by severity for each kind of issue there is extensive documentation. This greatly reduces the time to get started with security audits on software projects. It also helps the auditor to find out which software components should get the highest priority when fixing security flaws. These findings are supported by Shilling [3]

Future research should point out if and to what extend projects are adopting forms of static analysis for finding security flaws and also if there is a correlation between the use of static analysis and the number of bugs found during the lifecycle of a software project.

REFERENCES

[1] **Brian V. Chess.** *Improving Computer Security using Extended Static Checking.* IEEE Symposium on Security and Privacy Proceedings, 2002: 160 -173

[2] **David Evans and David Larochelle.** *Improving Security Using Extensible Lightweight Static Analysis.* IEEE Software, Jan/Feb 2002: 42-51

[3] **Walter W. Schilling and Mansoor Alam.** *Integrate static analysis into software development.* EE Times-India, Nov 2006

[4] **Yichen Xie and Alex Aiken.** *Saturn: A SAT-based Tool for Bug Detection.* ACM Transaction on Programming Languages and Systems (TOPLAS),May 2007, Volume 29, Issue 3

[5] **Common Vulnerabilities and Exposures.** http://cve.mitre.org (December 2008).

[6] **Fortify. "Source Code Analyzer." 2008.** http://www.fortify.com/products/detect/in_development.jsp (December 2008)

[7] **David A. Wheeler, "*SLOCCount.*"** http://www.dwheeler.com/sloccount/ (12 December 2008)

[8] **"TIME(1)." 14-11-2008** http://www.kernel.org/doc/man-pages/online/ pages/man1/time.1.html (December 2008)

[9] **"MAKE(1)." 22-08-1989** http://unixhelp.ed.ac.uk/CGI/man-cgi?make (12 December 2008)

Opponent Modelling for Limit Poker

F. L. de Vries and G. Veenstra

Abstract— Opponent modelling is an interesting challenge in poker. In a poker game opponent modelling can be applied to predict the hand strength or to the next action of an opponent. These can be used to improve the game of computer programs which try to play poker. This paper will focus on the finding of an opponent model which can predict the next action of a human poker player. This is done using a neural network, with the goal of finding a kind of ”default” (human) poker player.

Index Terms—Opponent Modelling, Limit Poker, Neural Network.

1 Introduction

For humans a game is relatively easy to learn. Just by playing a game several times we can become very good in mastering the game. On the other hand learning a game is a rather difficult process for a computer. The computer has to play the game more often than humans and it needs a lot more time to learn game strategies, using existing machine learning techniques. For this reason games are very interesting for (artificial intelligence) research.

When we look at games there are differences in complexity of the game play. Currently, one of the most complex board games is GO. Poker is also considered to be very complex. The game of poker requires anticipation of the opponent's play in order to use a good strategy. Even for most humans this is a difficult task.

Computers can anticipate on an opponent's play using opponent modelling (OM). The different techniques used for opponent modelling try to discover the non-observable properties in a poker game. Furthermore, skills like statistics for hand evaluation and risk management also very useful for good poker play. More important, if you know what your opponent is going to do, you can anticipate and use this to your advantage.

For our research we studied two different opponent modelling implementations, respectively opponent modelling using hand strength and hand potential and opponent modelling using an artificial neural network (NN). Our focus will be on opponent modelling using the neural network to predict actions of the players.

2 Texas Hold'em

There are several game varieties in the world of poker. This section will explain the popular variety Texas Hold'em which is used for this research. Texas Hold'em has two different betting styles, Limit Hold'em and No-Limit Hold'em. We will use Limit Hold'em which has in contrast to No-Limit restricted bet-sizes.

Each hand is divided in four betting rounds. In the first round also called 'pre-flop' the player is dealt two hole cards face down. The second round, called the 'flop', three board cards are dealt face up. In the third round, the 'turn', one board card is dealt face up. The final round is called the 'river' and again a single board card is dealt face up. In each round the player is able to fold, check/call or raise their hand. By folding the hand the player throws away his cards, with a check/call the player accepts the bet/raise from other players.

The board cards are community cards and players use these cards in combination with their hole cards to form the best five-card poker hand, see figure 1 for all poker hand ranks. The player who formed the best five-card poker hand will win in the showdown. Another way to win is when there is only one player remaining and all the other players have folded.

3 Requirements for a pokerplayer

There are several requirements needed to become a world-class poker player. In order to make a competing pokerbot these key requirements are used. Billings et al. (1998) have identified most of the key requirements and implemented some as software modules in the architecture of the pokerbot Loki. The following requirements are taken from [6].

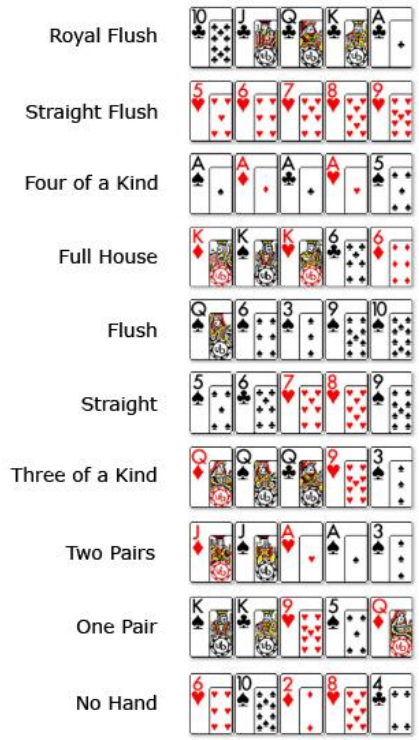


Fig. 1. Poker hand ranks. Taken from [http://www.poker5land.com/jpg/poker5s.jpg]

Hand strength is to calculate the strength or rank of a hand. Hand strength calculation is done in many different ways. The most simplest calculator only uses the pre-flop cards as information in order to calculate the hand strength. More advanced calculators use information like the board cards, amount of players and the position of players at the table. With this kind of information the hand strength is more accurate, furthermore, the possibility to calculate the hand strength of the opponent.

Hand potential is an important requirement when the strength of a hand is very low. Low hand strength pre-flop cards can increase potential on the flop when three board cards are dealt. Especially when a player is aiming for a straight or possible flush. In a case when a player is holding two cards of spades and the flop brings two more spades the hand strength will be very low. In contrast, the hand potential is very high because with one more spades on the board the player will hit the flush of spades. On the other hand when a player holds two tens and thus, a high hand strength, and the same flop comes up, the hand potential will drop because of a possible flush draw for the opponents.

Development of certain strategies are used to determine whether to

fold, call/check, or bet/raise in any given situation. Successful strategies are developed with hand strength, hand potential and especially opponent modelling.

Bluffing is trying to convince the opponent you have the best hand. This way you are able to 'steel' the pot with a low hand strength. Bluffing a low hand strength is considered difficult for human players. Human players need nerves of steel in order to bluff because when an opponent catches you bluffing you will probably lose the pot. Bluffing can be very successful combined with hand strength and hand potential information.

Unpredictability is important for you in order to keep the opponent guessing what strategy you are playing.

Opponent modelling is essential to achieve high performance in a poker game. Opponent modelling allows you to predict what action an opponent will make in any given situation.

4 State of the Art

The University of Alberta poker research group developed two poker bots (i.e., Poki [3] and Loki [5]). Both poker bots are implemented using opponent modelling. The research group is trying to defeat some of the best human poker players in Las Vegas using these poker bots [1]. In figure 2 we illustrate the architecture of the Loki pokerbot. The components and data structures are represented by the rectangles and rounded rectangles. Ovals correspond to actions and arrows reveal the dataflow. The Triple Generator is considered the beating heart of this

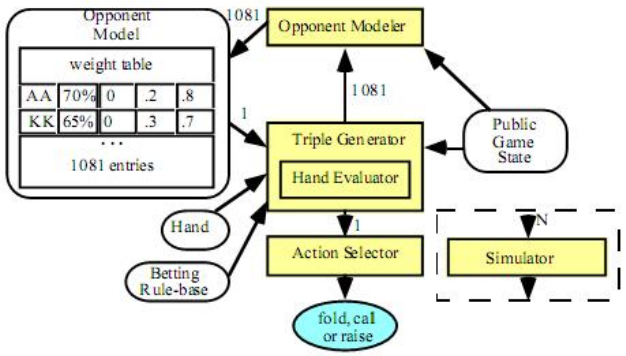


Fig. 2. Architecture Loki. Taken from [6]

architecture. This components gathers information like hand evaluation, game state and opponent modelling from other components in order to generate probability triples. A probability triple is an ordered triple of values, $PT = [f, c, r]$, such that $f + c + r = 1.0$. Each time it is Loki's turn to act the distribution gives the probability in order to select fold, call or raise in any given situation.

5 Method

There are two different reasons for using opponent modelling in poker. The first reason is to predict the hand strength and hand potential of the opponent [5].

The second reason is to predict what action an opponent will perform in any given game state. This form of opponent modelling can be done using an artificial neural network (with inputs like 'pot odds' and 'bets to call'). [8]

We develop a computer program with a neural network (NN) using an implementation idea used in the paper of Davidson (1999). [8]

The neural network will be trained using real data gathered from human players. We hope to see that after some training this neural network will be able to predict other human players.

The NN will be trained and optimized using a training dataset and an experimental dataset will be used to test if the NN is actually able to predict human actions in a poker game.

An overview of the procedure can be seen in figure 3. Each cycle we feed the NN the training dataset and after we freeze the NN we see how good the NN performs by measuring the average error of the experimental dataset.

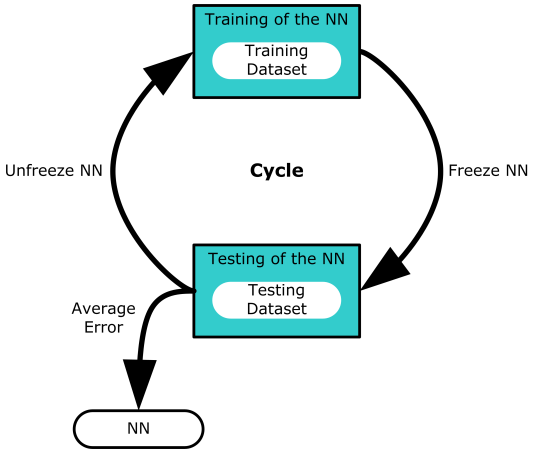


Fig. 3. Methode overview.

5.1 Opponent Modelling using Hand strength and Hand potential

This section will describe the first attempt of opponent modelling implemented in the pokerbot Loki. Billings et al. (1998) used enumeration techniques in order to calculate the hand strength and hand potential. Hand strength and hand potential are used to assess the quality of a hand. The assessment of the hand was in the first place used for Loki's own betting strategy and later on used to assess the opponent's hand.

Hand strength for Loki's own betting strategy is calculated as follows. Suppose the pre-flop cards are an ace of diamonds and a queen of clubs. Initially this is a good hand strength. But when the flop is dealt and shows a three of hearts, four of clubs and a jack of hearts most players would fold. To estimate the hand strength for this example Billings et al. (1998) developed an enumeration algorithm which is shown in figure 4. The algorithm returns the hand strength given any pre-flop cards and board cards. 47 cards (total of 52 cards - (two pre-flop cards + three flop cards)) remain unknown. Better hands are three of a kind, two pair, one pair or when an opponent is holding a ace and king. That makes a total of 444 possible hands that are better. Equal hands, which are ace and queens of other suits, are a total of 9 possible hands. A total of 628 possible hands are worse. For this hand the algorithm will return a 58.5% chance of winning the pot over a random hand. But this is only with one opponent, with five opponent's the algorithm returns $0.585^5 = 0.069$ which is only $6,9\%$ chance of winning the pot over a random hand.[5].

```
HandStrength(ourcards,boardcards)
{ ahead = tied = behind = 0
  ourrank = Rank(ourcards,boardcards)
  /* Consider all two card combinations of    */
  /* the remaining cards.                     */
  for each case(oppcards)
  { opprank = Rank(oppcards,boardcards)
    if(ourrank>opprank)          ahead += 1
    else if(ourrank=opprank)      tied += 1
    else /* < */                behind += 1
  }
  handstrength = (ahead+tied/2)
                 / (ahead+tied+behind)
  return(handstrength)
}
```

Fig. 4. Hand strenght algorithm. Taken from [5]

Hand potential is enumerated as followed. Again the same flop as in the example above, but now with a five and a two of hearts as pre-flop cards. This hand is considered very weak using the hand strength algorithm. But, the turn and river will bring two more cards on the board, and any heart, ace or six will give a flush or straight. And thus, this hand has a great potential to win the hand even though the hand strength is still low. So, what would be a good betting strategy in this state? In order to choose the right strategy, Billings et al. (1998) use a hand potential algorithm which is shown is figure 5. This algorithm, given any pre-flop and board cards, returns the positive potential (Ppot) and negative potential (Npot). Ppot is the probability to improve the best hand while the hand strength is low. Npot is the probability of falling behind while the hand strength is high. In this situation, all possible turn and river cards combinations from 45 cards, which make 990 possible combinations to calculate how many times the hand is ahead, behind or tied. Figure 6 shows the numbers of this calculation for the pre-flop cards, ace of diamonds and a queen of clubs, and a dealt flop with three of hearts, four of clubs and a jack of hearts. The figure shows that there are 91,981 ways to become ahead of the opponent, 1,036 ways to tie and 346,543 ways to stay behind the opponent. The algorithm returns for the Ppot a value of $0.208 \approx 21\%$ and for Npot $0.274 \approx 27\%$.[5]

```
HandPotential(ourcards,boardcards)
{ /* Hand potential array, each index repre-  */
  /* sents ahead, tied, and behind.           */
  integer array HP[3][3]   /* initialize to 0 */
  integer array HPTotal[3] /* initialize to 0 */

  ourrank = Rank(ourcards,boardcards)
  /* Consider all two card combinations of    */
  /* the remaining cards for the opponent.    */
  for each case(oppcards)
  { opprank = Rank(oppcards,boardcards)
    if(ourrank>opprank)          index = ahead
    else if(ourrank=opprank)     index = tied
    else /* < */                 index = behind
    HPTotal[index] += 1

    /* All possible board cards to come.      */
    for each case(turn,river)
    { /* Final 5-card board */
      board = [boardcards,turn,river]
      ourbest = Rank(ourcards,board)
      oppbest = Rank(oppcards,board)
      if(ourbest>oppbest)      HP[index][ahead]+=1
      else if(ourbest=oppbest) HP[index][tied]+=1
      else  /* < */           HP[index][behind]+=1
    }
  }
  /* Ppot: were behind but moved ahead.       */
  Ppot = (HP[behind][ahead]+HP[behind][tied]/2
                           +HP[tied][ahead]/2)
       / (HPTotal[behind]+HPTotal[tied])
  /* Npot: were ahead but fell behind.        */
  Npot = (HP[ahead][behind]+HP[tied][behind]/2
                           +HP[ahead][tied]/2)
       / (HPTotal[ahead]+HPTotal[tied])
  return(Ppot,Npot)
}
```

Fig. 5. Hand potential algorithm. Taken from [5]

5 Cards	7 Cards			
	Ahead	Tied	Behind	Sum
Ahead	449005	3211	169504	621720 = 628x990
Tied	0	8370	540	8910 = 9x990
Behind	91981	1036	346543	439560 = 444x990
Sum	540986	12617	516587	1070190 = 1081x990

Fig. 6. Hand potential calculations. Taken from [5]

Furthermore, these enumeration techniques, slightly modified, are able to create a model of the opponent (OM). A separate set of weights is assigned to each opponent. The initial weights s for every opponent are an array of weights s indexed with all possible pre-flop cards. When an opponent raises on the flop his weights s are increased for all possible strong hands, given the flop. For weak hands the weights s are decreased. After a few games played, the weights s represent a relative probability of the opponent's hand.

5.2 Opponent Modelling using Artificial Neural Networks

In the previous section we explained opponent modelling with hand strength and hand potential. Davidson (1999) states that the current (hand strength and hand potential) opponent modelling used in Loki is very crude. According to Davidson too little information which is available is used given every played game. Figure 7 shows which information Davidson used for his input nodes. [8]

#	type	description
1	Real	Immediate Pot Odds
2	Real	Bet Ratio: bets/(bets+calls)
3	Real	Pot Ratio: amount_in / pot_size
4	Boolean	Committed in this Round
5	Boolean	Bets-To-Call == 0
6	Boolean	Bets-To-Call == 1
7	Boolean	Bets-To-Call >= 2
8	Boolean	Stage == FLOP
9	Boolean	Stage == TURN
10	Boolean	Stage == RIVER
11	Boolean	Last-Bets-To-Call > 0
12	Boolean	Last-Action == BET/RAISE
13	Real	(#players Dealt-In) / 10
14	Real	(# Active Players) / 10
15	Real	(# Unacted Players) /10
16	Boolean	Flush Possible
17	Boolean	Ace on Board
18	Boolean	King on Board
19	Real	(#AKQ on Board) / (# Board Cards)

Fig. 7. Input nodes with boolean values of 0 or 1 and real values from 0 to 1. Taken from [8]

Before the network is able to predict an opponent's next action the network needs training. The training phase is done by feeding the network several log files with observation data. While the observations are fed to the network, the network is able to learn associations between the input information and the opponent's actions.

6 Implementation

We used the same neural network as Davidson (1999) describes in his paper about opponent modelling. The neural network is a standard feed forward network which consists out an input layer with 19 nodes, hidden layer with 4 nodes and an output layer with 3 nodes. Figure 8 shows a clear view of this neural network implementation. The red (negative) and black (positive) connections represent the weights s between the nodes. [8]

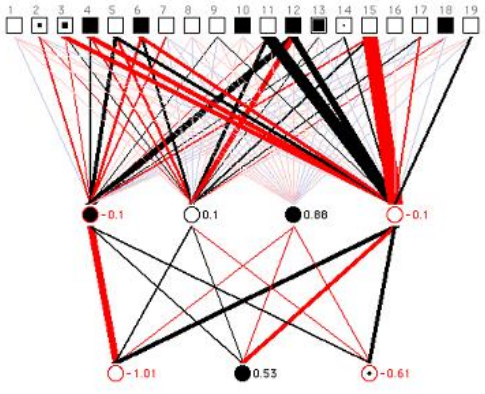


Fig. 8. Neural Network. Taken from [8]

The error for each output node is computed by subtracting the actual output from the desired output:

$$\forall output : e_j = (d_j - Y_j) \tag{1}$$

The average network error can be calculated by taking all the squared errors of the output layer and dividing them by the number of output nodes.

$$e_{average} = (e^2_{fold} + e^2_{call} + e^2_{raise}) * (1/3) \tag{2}$$

The error in the hidden nodes is computed by the following equation:

$$\forall hidden : e_j = \sum_{i=0}^{N} (w_i * e_i) * (a_j - 1) * a_j \tag{3}$$

where a_j is the output of this node, w_i is the weights of node i in the previous (output) layer, e_i is the error of node i in the previous (output) layer and N is the number of nodes in the previous (output) layer (N=3).

Neuron weights adjustments where made with the following equation:

$$\forall weightss : w_{ij} = w'_{ij} + (LR * e_j * X_i) \tag{4}$$

Where w_{ij} is the new adjusted weights , w'_{ij} the current weights . e_j is the computed error and X_i is the output of the neuron in the previous layer (node i). The initial weights s are randomly chosen between -0.5 and 0.5.

The value of the learning rate $LR = 0,01$. This is the part where we differ from the implementation of Davidson (1999). Davidson (1999) used an default learning rate of $0,4$, the average error (Equation 2) was used as learning rate depending which was smaller [8]. As a consequence, the implementation of Davidson (1999) will make bigger steps in the beginning of the learning phase and as the network converges the learning rate will decrease because of the dropping error. The reason we used another learning rate as Davidson (1999) is that we wanted a constant and relatively small learning rate. Despite the fact that the learning process of the NN is slower, the NN has less chance of getting stuck in local minimum. Getting stuck in local minimum means that the NN is training towards a solution that is not the global best solution.

$$LR = (0,01) \tag{5}$$

The precision is calculated as denoted in equation 6. It shows the percentage of all the correct predicted actions.

$$\begin{aligned} precision \quad = \quad & (fold_{correct} + call_{correct} + raise_{correct}) \\ & / total_{actions} \end{aligned} \tag{6}$$

7 Results

In the following sections we describe results of our experiment cycles. In the first experiment we used the same neural network (NN) implementation as Davidson (1999). We conducted another experiment to test the NN with more hidden nodes in order to increase our performance. At the end of this section we compare our results with Davidson (1999). The training dataset, used for each experiment exists of 504 players and 24227 actions. To test the NN a validation is done with the experiment dataset which exists of 371 players and 21358 actions.

In figure 9 we compare the results of the decreasing average error from both neural networks. We can conclude that the NN with 8 hidden nodes is performing slightly better than the NN with 4 hidden nodes. This is probably due to the fact the NN with 8 hidden nodes is able to learn more features. Because of the better results with 8 hidden nodes we only discuss this NN.

Furthermore in figure 9, after ten cycles the error is dropped from 0.29 to ≈ 0.20. The error stabilizes after approximately 100 cycles. The results of the experiment dataset are shown in table 1.

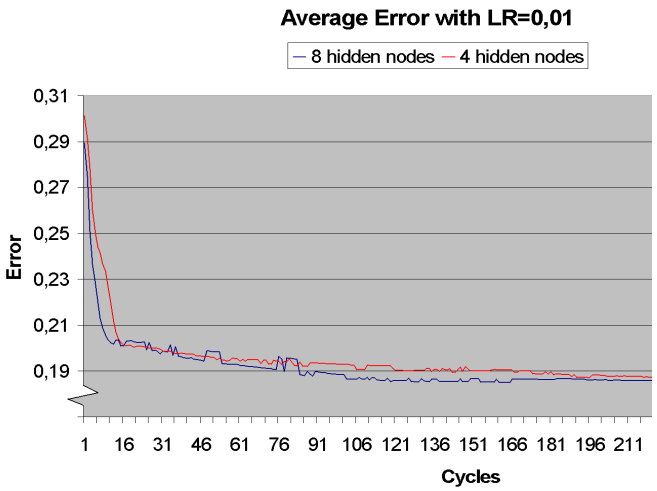


Fig. 9. Training curve with 4 and 8 hidden nodes

	F	C	R	
F	**322**	406	58	786
C	1889	**12731**	1521	16141
R	101	366	**3964**	4431
	2312	13503	5543	21358

Table 1. Results (8 hidden nodes).

In table 2 we show the confusion matrix of the predictions of an action and the actual actions performed by a player. The precision of our NN is $79,68\%$ which means that $79,68\%$ of all prediction where correct. The confusion matrix also reveals that the NN struggles to predict correct fold actions. Because when we compare the percentages, $1,51\%$ correct fold predictions of a total of $10,82\%$ actual folds, the accuracy for the fold $(1.51/10.82) \approx 14\%$ seems rather low compared to the accuracy of the call $(59.61/63.22) \approx 94\%$ and raise $(18.56/25.95) \approx 72\%$ actions.

prediction	actual action Fold	Call	Raise	
F	**1,51%**	1,90%	0,27%	3,68%
C	8,84%	**59,61%**	7,12%	75,57%
R	0,47%	1,71%	**18,56%**	20,75%
	10,82%	63,22%	25,95%	
Precision: **79,68%**				

Table 2. Confusion matrix (8 hidden nodes).

The results of Davidson (1999) are shown table 3. There are a few differences compared to our results. First of all, Davidson (1999) has a higher precision, $85,60\%$ compared to $79,68\%$. Secondly, we noticed that Davidson (1999) has very good performance on predicting the fold actions.

prediction	actual action Fold	Call	Raise	
F	**13,00%**	0,03%	0,03%	13,60%
C	0,0%	**58,40%**	3,30%	61,80%
R	0,0%	10,50%	**14,10%**	24,70%
	13,0%	69,30%	17,70%	
Precision: **85,60%**				

Table 3. Confusion matrix (Davidson (1999), taken from [8]).

8 Conclusion

We did find a neural network (NN) that can be used as a starting point for modelling individual opponents. This concept could be deployed in an artificial poker player in combination with other poker playing methods.

Despite the fact that the implementation of Davidson (1999) has a higher precision, we are satisfied with our results. Due to time constraints we could not further improve our results. This especially yields for the prediction of the fold actions.

9 Discussion

There still should be done some research into using the prediction of the next action of a player. One possible way of using these results is by generating a clear probability triple for the next possible actions. Furthermore, when these results are actually used, it could also have a negative side effect in that other players are able to adjust their strategies.

Divide the NN in three separate neural networks could also improve the precision. Thus, training one NN for each action separately.

Maybe in combination with an action-frequency predictor some improvements could be made. Predictions of the action-frequency predictor could be used in combination with the predictions of the neural network. Confusion matrices could reveal the correlations between the both predictions.

10 Future work

Future work could include expanding the default model towards multiple opponent models based on the strategy used by a player (e.g. an aggressive/tight strategy). There is also the possibility of competing neural networks. With the use of genetic algorithms the best neural network is selected. This could result in the selection of the best prediction model per poker player.

Future work could even focus on more subtle aspects of the poker play (e.g. time taken before acting).

"There is no shortage of good ideas to investigate; only a shortage of time and resources.", Billings et al. [6]

References

[1] The University of Alberta computer poker research group. Website: http://games.cs.ualberta.ca/poker/.

[2] R. Berteig. Basic Concepts for Neural Networks. Website: http://www.cheshireeng.com/Neuralyst/nnbg.htm, October 2003.

[3] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[4] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Poker as a testbed for ai research. 1997.

[5] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *Proc. AAAI-98, Madison, WI (1998)*, pages 493–499, 1998.

[6] D. Billings, L. Pea, J. Schaeffer, and D. Szafron. Learning to play strong poker. *Machines That Learn To Play Games*, pages 225–242, 2001.

[7] D. Billings, L. Pena, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *Proc. AAAI-99, Orlando, FL (1999)*, pages 697–703, 1999.

[8] A. Davidson. Using artificial neural networks to model opponents in texas hold 'em. *Unpublished manuscript; http://spaz.ca/aaron/poker/nnpoker.pdf*, November 1999.

Challenges of systems biology: Pathways and Visualisation

Eric Begue, Bram Leemburg

Abstract— In the field of Bioinformatics, scientists use visualisation techniques to analyse pathways, i.e. representations of sequences of biological reactions. As these pathways can be complex, efficient visualisation techniques help to understand the role of substances within the pathways. Graph representations are commonly used in pathways visualization for representing entities and their relations as nodes and edges. For life scientists graph representations focus much on the relational structure of entities. The actual biological meaning is lost when the meaning of the entity relations are not visible. To understand the problems faced by life scientists, the properties of the pathways have to be analysed. In a study on requirements [1] it is shown that life scientists lack the right tooling to support their research. This paper has the purpose of providing an overview of the state of the art in the pathway research field. The intention is to provide further research directions to computer scientists contributing to the field. Temporal information is an important aspect for pathway visualisation systems according to a study [1] based on interviews of life scientists about their usages and needs of such systems. In our paper we introduce a technique used in the software development domain: the timeline visualization. That is used in Solid Trend Analyser - a software development tool used to visualize the evolution of numerous source code files on depositories. As Solid Trend Analyser is highly time visualisation oriented, we provide rationale and motivation on how this representation can be adapted in pathway visualisation. The expressive power of this visualization technique is evaluated. For this we compare the Solid Trend Analyser to existing attempts to integrate temporal data in pathway visualisation tools.

Index Terms—Bioinformatic, Pathway, information visualisation

1 Introduction

Biological pathways are diagrams that describe interactions within living system. They model how biological molecules interact to accomplish a biological function and to respond to environmental stimuli. Life scientists use pathways as a communication language that conveys information about biological processes.

Life scientists are the main users of biological pathways. They use these diagrams in different context of activities: integrating results from literature, capturing empirical results, sharing current understanding and even simulating processes. A common goal of research in the life sciences is to develop an ever-broadening library of pathway models for biological processes of many different organisms. In the field, specialized scientists working in small teams typically conduct research. Although these specialists may know all the details on a particular pathway, the pathways are not islands. Participants in one pathway may also affect other pathways, essential for example to analyse side effects of medication.

Over the last years life scientists have gathered huge quantities of data about molecular processes within cells and at higher levels. Most of this data has been revealed by genomic research [8]. Intricate processes of genes affecting one another, regulation processes and differences in the expression of genes have been found. Also other research on neurobiology revealed how neurons use different processes to relay signals within the body. Understanding these pathways scientists is essential for development of medication and to increase overall knowledge of all living creatures. In order to understand the processes, life scientists need to communicate their knowledge and findings and specify the relations of the processes.

Pathways are extremely diverse. They can be very specific, highly abstract, sketchy or rigorous. Figure 1 shows some examples of different pathway diagrams.

In biochemical research, huge amount of data are produced by experiments such as micro-arrays or gathered by several teams in their collaborative work. Bio informatics provides tools for building, storing, sharing and analysing biological pathways. An important technique for analysing biological pathway is visualization.

•*Eric Begue is with RuG university, E-Mail: ericbeg@gmail.com.*
•*Bram Leemburg is with RuG university, E-Mail: ericbeg@gmail.com.*

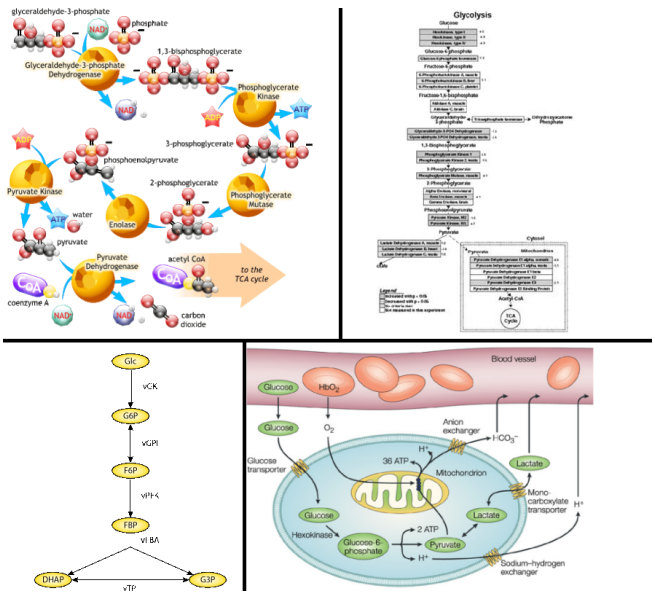


Figure 1 - different illustrations of the same pathway (glycolysis), note the difference in details and descriptions

For helping life scientists in their research, several visualization systems exist and offer several features. A results of a study [1], based on interviews of live scientists about their usage of pathways visualization systems, provides a list of thirteen requirements categorized in three groups: (1) Pathway assembly, concerning the construction of pathways (2) Information overlay, concerning information about properties of pathways elements or additional information such as sources (3) Pathways analysis, concerning the features for analysing pathways, e.g. Overview, levels of abstraction, specialized view or focus.

From the analysis of the visualisation tools was concluded that the problem with existing techniques are focussed on only one purpose. There is no tool that implements all the requirements that

the scientists have. Therefore scientists are sceptical about the systems, because they are forced to learn many different systems to access the required functionality. An interesting approach is the attempt to provide an integration framework [3] of the various techniques. In this paper we give an understanding in the difficult aspects of visualising pathways and provide a new interesting research topic that may improve the ability to depict the dynamics of the processes involved. For this the domain-specific properties have to be analysed.

1.1 Problem Statement

This paper is written as introduction for computer scientists into the field of biological pathway visualisation. It provides an analysis of the challenges in the field, based on literature research. Also it provides an overview of the tools available and their ability to support life scientists. We attempt to provide motivation and research direction to visualisation designers.

2 Exposition

First we provide the relevant biological information on pathways and visualization of the pathways. Secondly, we show the requirements on visualization tools and provide insight in the state of the art of visualization tools in the field. Finally research directions are suggested and the general shortcomings and discussed.

2.1 Biological Pathways

To understand the intricacies of visualization of Biological Pathways, first a basic understanding of the underlying pathways is required. A biological pathway describes how an organism performs a biological function [5]. A biological function is an observable physical or biochemical property of an organism. The expression of functionality by an organism is called phenotype. A biological pathway describes the substances and reactions causing the expression of the phenotype of an organism. For example, the human body can break down hydrocarbon molecules food to get energy. This is an observable property that humans posses. This property is caused by what is called the metabolic pathway. It consists of many biochemical entities throughout the body that break down molecules and carry energy to where it is required.

Pathways are comprised of a sequence of small steps in which biochemical entities get input substances and have certain outputs. Describing pathways helps life scientists in many ways. It helps them to express their understanding of organisms. It provides a common language to communicate their understanding. Also it provides the link between functionality and the actual physical entities present within organisms. This enables is useful to understand and simulate the effects of chemicals on organisms, fundamental for drug research, discovery of deficiencies, understanding the functions of genes and numerous other biochemical applications.

2.1.1 Properties of Pathways

As mentioned before pathways are defined as the sequence of biochemical reactions causing a certain observable property. However the biochemical entities are not just responsible for one of these properties. The entities are therefore often present or related to multiple pathways, causing the pathways to be related. The pathway for breaking down sugar; the glycolysis, yields some energy that is passed along, but also outputs smaller molecules which are fed to the Krebs-cycle pathway. The Krebs-cycle can also get input from the break-down of other substances, such as fats, caused by other pathways. This process is illustrated in figure 2. Other than these type of input/output relations and multiple functions of entities, parts of a pathway may be controlled by other pathways or have some other type of relation [6].

The average size of pathways that are under research is between 50 to 100 entities. This number rapidly increases when related pathways have to be taken into account. This is a problem for scientists, because it makes it hard to understand pathways and depict them.

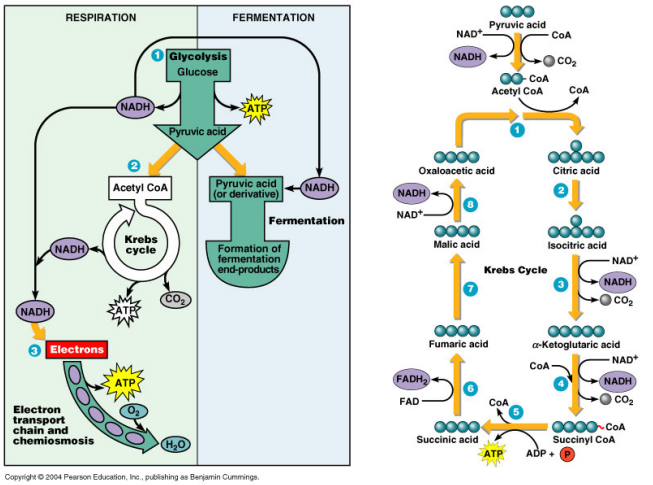


Figure 2 – metabolic pathway interactions from [2]

left: interactions between the glycolycis and other pathways, right: detailed view of the krebs-cycle

In the literature [3] pathways are divided on different levels. There are pathways on the genomic, transcriptomic, proteomic and metabolic levels. The genomic level is about genes and how these are affected by one another and external entities. The transcriptomic level is about the creation of proteins from the genetic code. The proteomic level is about the functioning of proteins. And finally the metabolic level is about how the processes are provided with energy. As becomes immediately evident, these levels are intertwined, for example, at all levels energy is required, for example for duplicating genetic material. The levels themselves can also be somewhat hierarchically structured, for example the glycolysis and krebs-cycle are parts of the metabolic pathway.

For life scientists, another important aspect is where the processes take place and when. The question of where can be specified as the type cells or organs, but also to the extent of the location within the cell. As for the question of when; the order and dependencies between reactions is particularly important. However, experimental data, such as data gained from micro arrays is also inherently time-dependent. Having insight in these dynamic properties is equally important to scientists as seeing the static structure of the pathway.

2.1.2 Pathway representations

Describing pathways is not easily accomplished. We have explained already several properties of the pathways that make this evident. In bio informatics, pathways are classically represented using graphs. Simple graphs, however are not able to present all the information. Newer pathway databases use ontology languages [7] to describe the pathways. Although this technique is more capable, no consensus on the vocabulary is established yet. Important to realize is also that pathway research can give incomplete, uncertain and contradicting results. The pathway descriptions have to reflect this so that life scientists can proceed in their research.

To summarise, we have explained the basic concept of pathways and shown the traits of pathways that make it hard for scientists to get an understanding of the processes at the cellular level. These traits are the relations between pathways, the size of pathways, the

Table 1 - High level pathway visualisation requirements

Categories	Requirements	Tasks
Pathway assembly	1. Construct and Update	Collect and link pathways from multiple resources
	2. Context	Provide information about pathways
	3. Uncertainty	Maintain alternate hypotheses and information reliability
	4. Collaboration	Enable group work
Information overlay	5. Node and edge representation	Details about network entities and interactions
	6. Source	Details about source resources
	7. Spatial information	Physical locations of pathway entities in the cell
	8. Temporal information	Time-related properties
	9. High-throughput data	Expression data from high-throughput experiments
Pathway analysis	10. Overview	Comprehend large or multiple pathways
	11. Inter-connectivity	Intra- and inter-pathway effects of entities on each other
	12. Multi-scale	Relate networks at different levels of abstraction
	13. Notebook	Track accumulated research information

The requirements are grouped into three main categories: pathway assembly, information overlay, and pathway analysis.

intertwined levels of pathways, and the temporal and spatial properties. Also the uncertainty of pathway data needs to be accounted for in some way. Next we will describe the approaches taken to provide visual aids to scientists in the field.

2.2 Biological Pathway Visualisation

In the biological research domain, Information Visualization technology enable scientists to use pathway as a focal point to integrate diverse related information, such as literature citations, research notes, and experimental data.

In the recent years, large amount of experimental data are generated from high-throughput capture technology. Theses experiments provide data about thousand entities. In the context of pathway diagrams, all this data must to be analysed to enable biologists to derive new knowledge about the underlying biological processes and to improve the current pathway models. The combination of numerous entities and large amount of related information increase the complexity of pathway diagrams. Some visualisation tools ,presented in the next section, provide features that can abstract that complexity.

In biological research, the increasing importance exploratory pathway analysis corresponds to a major shift beyond the reductionist scientific process, which rigorously examines each interactions of biological molecules, towards system-level science, which explore entire systems of many biological molecules. System-level science highlight that the whole is greater than the sum of the parts. A challenging goal for pathways is to try to convey complex global functionality, interconnections with other pathways, and their dynamic behavior. [1]

Visual representations are necessary to facilitate exploratory analysis of complex pathways. Pathways are typically represented as network diagrams. Some pathway diagrams are very pictorial such as found in the literature, e.g. Biocarta, while others are very basic, made of black line and simple circle, see figure 1. Visualisation systems interactively generate pathways diagrams in respect of the user interest. The fish-eye zoom feature of GScope, for example, gives the possibility to the user give more importance to some part of the pathway by displaying entities in the centre of the screen bigger than those that are on the border.

2.3 Overview of Visualisation tools

To give an overview of the tools that exist in the field it would be impossible to list all features of all the available tools. Instead we highlight some of the approaches provided by tools to accommodate the requirements from [1]. Here the focus is on the visualisation itself, not so much on the pathway assembly. As can be seen in Table 2, the features of the existing tools are spread scarcely.

2.3.1 Relating pathways

To provide overview and interconnectivity of pathways, some approaches can be found in existing tools. These approaches are related to the Pathway analysis requirements described earlier.

Figure 3 - GenMapp, pathways are linked through a hierarchy

In GenMapp, as depicted in figure 3, the pathways are related using a tree structure. Users can browse the tree to get related pathways. This functionality is not as powerful as being able to depict all related pathways and showing the type of relation. As stated in the introduction, relations between pathways cannot be mapped onto a strict hierarchy.

2.3.2 Semantic zooming

Related to the features for navigating pathways is the semantic zooming. This feature is also found in dynamic road map applications. It is the ability of the tool to hide details as the scale increases. This could also create overviews of inter-pathway relationships. In figure 4 the approach was implemented to size-up a portion of the diagram to reveal more details.

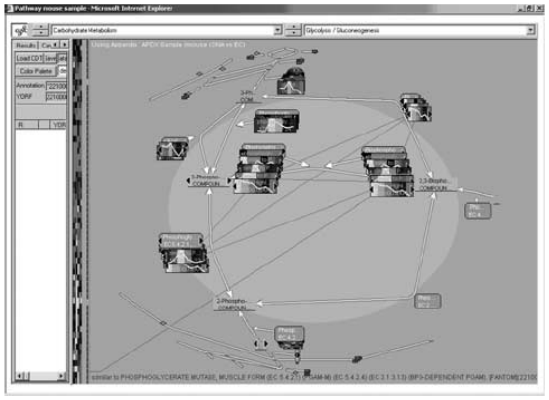


Figure 4 - GScope, a tool with a fish-eye zoom feature

In figure 5 an experimental approach is shown that allows multiple detail levels to be mapped on each other, using meta-graph representations [4]. This technique allows users to zoom in on sub-graphs of their interest. In the article on meta-graphs [4] this is formulated as *"The popular website Google Maps, for example, implements a familiar form of semantic magnification in which geographic lables ranging from street names to political boundaries are automatically set to adjust for changes in zoom level. Semantic zooming makes use of the viewing context to decide, in real time, what kind of information to include."* This is a more powerful feature than the fisheye, because it allows more different levels of detail. It should be noted that the detail levels are not the same as the hierarchy of pathways described earlier. As can be seen in figure 5, all levels are metabolic and show the same type of interactions, only the level of detail differs, as can be selected by the user. All variants of a protein complex can be depicted as one node or multiple as demanded.

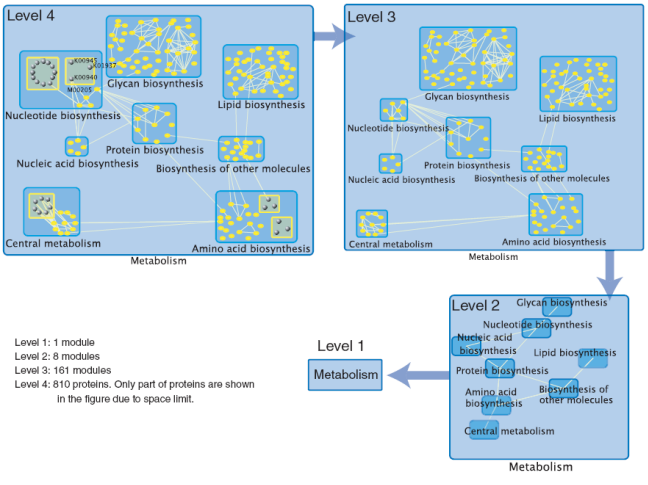


Figure 5 - Multiple levels of detail

This technique could also be a powerful way of linking pathways and creating pathway overviews, this technique is not fully implemented in existing tools. Managing the lay-out of these overviews and allowing user interactivity are features that remain a challenge.

2.3.3 Spatial properties

A problem with the automated pathway visualisation tools is that spatial properties of the pathway are not depicted well. Text book type of pictures often show drawings of the involvement of organelles and cell membranes. An attempt to visualise the location of steps of the pathway is shown in figure 6. Different areas are separated into different windows. This approach is not as visual as the textbook pictures but functional distinctions can be made.

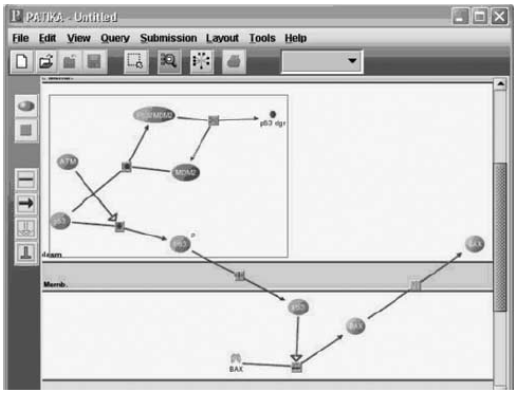


Figure 6 - Patika's approach to separating locations

2.3.4 Timelines

Perhaps least addressed by current tools is visualisation of the temporal dependencies and related data (such as expression values). In Figure 7 this is attempted by integrating function views into the tool. This approach is limited in the sense that the user must make the link with the static structure of the pathway and the visualisation cannot express this itself.

2.4 Research suggestions

Having given an overview of the features offered by current tools, we now try to give some directions for conceptual visualisation techniques that may yield improvements. The implementation and verification of the applicability of these techniques falls outside the scope of this paper.

2.4.1 Timeline based visualisation

In the field of visualisation of software, an interesting tool is developed by Solid Source; the Solid Trend Analyser. This tool is used to analyse huge repositories of source code. The problems faced in the software engineering are similar in nature to the biological pathway visualisation, the data also has complex relations, is time-dependent and is to large to effectively analyse using conventional graph layouts. In fact in the literature [1], system biology is compared with computer scientists attempting to reverse engineer a complex system. The solution developed here is to display the entities as timelines and apply color overlays to indicate metrics, such as the size of the entity, and relations, such as the participating author. Our concept is to reuse this technique and apply it to pathway visualisation.

The Solid Trend Analyser's main view is the entity lifeline as shown in figure 8. The program can apply color schemes to the

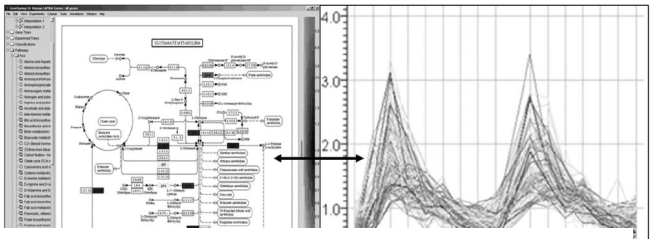


Figure 7 - CytoScape, multiple views allow micro-array data to be shown over time

entities, based on metrics or relations. The entities can be grouped and filtered based on these color distinctions. The timeline visualization can provide good insight for data analysis, not just capable of representing the relations, but also displaying quantitative metrics, such as the size of the entity or the amount of involved reactions in the same way.

The method is based primarily on the time dependent aspect of the data, not properly addressed by graphs. For this reason it can serve as a data driven analysis tool, in other words be used to visualise experiment output.

Another strength of this technique is the ability to represent extremely large amounts of entities; the technique has very good scaling capabilities. Unlike with a graph, no connections between the entities are indicated; instead the relational property is indicated with a color.

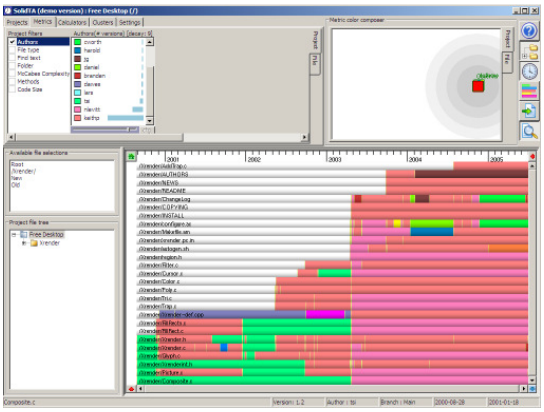


Figure 8 - the Solid Trend Analyser's timeline visualisation

3 Discussion

Within the SolidTA tool multiple levels are hard to separate. Strict hierarchies can be visualised, originally performed for showing filename hierarchies in the software repositories. The tool does have the ability to use multiple color overlays and filters, which may satisfy this requirement.

The most difficult aspect is to depict the input and output behaviour of pathway entities. The dependencies have to be clear while the involved entities are positioned on different lines. Dynamic grouping of the entities based on their dependencies is required, but will be difficult to do in a clear way. Using color markers, transition lines and some shearing of the timeline will have to be researched to satisfy this.

A problem that remains is that the color schemes used mainly in SolidTA do not scale so well. The eye can only distinguish a limited amount of different colors. This makes it hard to visualize all involvements in some relations.

Even when the afore mentioned challenges are overcome, this technique still will not solve the main problem, integration of the requirements into a single framework [3]. It may however bring a better tool for visualising the dynamics of pathways.

4 Conclusion

We have tried to give an overview of biological pathway visualisation to fellow computer scientists to give insight in the state of the art and to analyse the problems faced in the field. Understanding the properties of pathways is crucial to realise the demands on pathway visualisations. Pathways consist of large amounts of biochemical reactions having complex interactions. Biological background information such as the type of interaction, the location and the timing aspects are important for researchers in the field. Pathway visualisations need to show the overview to allow system-level studies, because "*The whole is greater than the sum of the parts*" [1]. Scientists are interested in discovering the processes behind the physiological effect.

Existing visualisation tools are as numerous as they are limited. To perform all the functionality that a life scientist needs, as shown in table 1 a large number of tools is required. The system that exist offer useful features for helping life scientists in their activities. But, according to [1] life scientists skeptical about the biological value of current pathway visualisations. The reluctance of using the currents pathway visualisation systems remains in two intrinsically related points. Firstly, the features needed by life scientists are scattered into several systems, table 2 highlights this point. Secondly, the time required to learn several systems is too important compared to the benefits gained by mastering theses systems. Integrating the different tools and representations will be highly beneficial to the user community [3].

Existing tools are mainly focused on visualising the pathway as a graph, leaving out important biological details. To support the visualisation of temporal details more research is required on similar tools used in software system analysis. A promising tool for visualising software changes over time could be adopted to visualise the temporal aspect of pathways.

References

[1] Purvi Saraiya, Chris North1, Karen Duca. Visualizing biological pathways: requirements analysis, systems evaluation and research agenda. Information Visualization (2005), 1–15. 2005 Palgrave Macmillan Ltd

[2] Pearson web courses on metabolism, http://classes.midlandstech.edu/carterp/Courses/bio225/chap05/

[3] Jos B. T. M. Roerdink et al, MOVE: A Multi-level Ontology-based Visualization and Exploration framework for genomic networks (2006), in Silico Biology 7, 0004

[4] Zhenjun Hu et al, Towards zoomable multidimensional maps of the cell (2007), Nature Biotechnology vol 25, 5

[5] Dee Unglaub Silverthorn, Human Physiology, Pearson 2007 fourth edition, pages 1-10, chapter 1 Introduction to physiology

[6] Dee Unglaub Silverthorn, Human Physiology, Pearson 2007 fourth edition, pages 101-108, chapter 4 Energy and Cellular Metabolism

[7] Demir, E. *et al*. An ontology for collaborative construction and analysis of cellular pathways. Bioinformatics 20, 349–356 (2004)

[8] Bruce Alberts et al, Molecular Biology of the Cell, Garland Science 2002, fourth edition, pages 25-26

APPENDIX

Table 2 – Requirements mapped onto the existing tool

Requirements	Approaches	SYSTEMS																						
		BIND	BioCarta	Biological Story Editor	Cytoscape	EcoCyc	GenePath	GeneSpring	GeneSys	GENEW	GENIES	GenMAPP	GScope	KEGG	Knowledge Editor	MapMan	OmniViz	Pathway Editor	PathwayAssist	PathwayFinder	Patika	PubGene	STKE	Unipath	Vector PathBlazer
R1	Reference	x	x			x								x									x		
	Pathway editor tools														x			x						x	
	Construct pathways using NLP algorithms from literature databases										x						x		x	x		x			x
	Construct pathways from microarray data						x		x	x															
	NLP algorithms to update local database																		x						
	update database manually																				x				
	Update pathway manually				x							x													
R2	Attach notes				x							x							x						
R3	Manipulate node and edges properties				x							x													
R4	Facilitate sharing across group members			x													x								
R5	Manipulate node and edges visual properties				x							x	x												
	Provides shape for different types of nodes																			x	x			x	
R6	Attach source information on nodes and edges				x							x							x						
R7	Provides different shapes to show different cellular location											x													
	Manipulate node properties and used fixed layout				x							x							x				x		
	Divide visualization in different areas																				x				
R8	Manipulate edges length, or layout pathway elements in the order they react				x							x							x						x
	Animations																						x		
R9	Overlay data on node (using color), one condition at a time				x							x							x						
	Embedded views, for multiple conditions												x												
	Multiple linked views, for multiple conditions							x																	
	Visualizations for a functional group															x									
	Automatically infer relationships between entities from data						x																		
	Overlaying replicates											x													
R10	Functional groups													x		x									
	Zooming				x																				
	Fish-eye views												x												
R11	Up-down cascades												x												
	Query pathways																		x		x				
R12	Chromosome location + pathways							x																	
R13	Attach notes to node and edges				x							x													
	Build stories about pathway elements			x																					

A discussion of secret information comparison methods

Pieter Noordhuis Harry de Boer

Abstract—This paper presents an overview of some classical secret information comparison methods present in literature. While a good overview is presented by Fagin, Noar and Winkler, they do not give a theoretical background. We provide an overview that does not only present the methods but also provides some theoretical background. Also a more recent method by Teepe is discussed.

1 Introduction

Some secrets are best kept to yourself. However, knowing that someone knows the same secret as you can be important. Suppose two persons claim to know a password to a secret society, but both do not know whether the other is indeed a member. They want to be sure each knows the password before discussing their secret activities. The problem is determining that both secrets are indeed the same without compromising them when they are not. A number of methods to compare secrets without giving them away on a mismatch are given in Fagin [2].

Several of these methods are fairly simple and can be executed easily, while others are too complex for the average user. Execution of a method to be used by two people without any computational device available should be very simple. This characteristic is less of an issue when computational power is available. However, they all rely on two parties knowing *which* secret they talk about (*with reference*). Teepe [5] proposes methods for comparing information out of a *set* of secrets so it is not known in advance for which secret possession is going to be proved (*without reference*). The latter methods rely on using padded one-way collision-free hash functions. Methods for proving possession of a secret from one party to another is not considered in this paper as we concentrate on mutually proving possession of secrets (i.e. comparison of secrets).

The key characteristic for these methods is that they do not reveal any information other than whether there is a match. Protocols used in these methods are therefore called *zero knowledge protocols*. These were first described in Goldwasser, Micali and Rackoff [4]. Characteristics such as security, simplicity and remoteness apply to all secret comparison methods. The importance of every characteristic depends on how the method is used. In example: remoteness is important when one party resides in Australia and the other in Europe, but is not an issue when they are in the same room.

In this paper we give a basic overview of the theoretical background used in secret information comparison methods. We give an overview of available methods, categorized in simple, third party and cryptographic methods. We discuss characteristics and flaws for the cryptographic methods as these are most interesting for computer science.

2 Theoretical background

This section provides preliminary knowledge for using and developing secret information comparison methods.

2.1 General characteristics

Not all methods are suitable for all applications. Therefore we list a set of characteristics that can help in selecting the right method that suits the requirements of the application.

- **Resolution** The goal of information comparison methods is to find out whether two parties have the same secret. This is what resolution embodies. Both party A and party B should find out if their secrets are the same. One might think that this characteristic will always be met. However, resolution is not always equally important when other characteristics are violated (privacy might be more important than resolution).
- **Leakage** When both parties followed the method of their choice faithfully, they should not get any knowledge other than whether their secrets are equal.
- **Security** In addition to leakage, security means that if one of the parties is trying to cheat, no knowledge is gained about the secret of the other party.
- **Privacy** Apart from the parties that want to know whether they have the same secrets, no one else should gain any knowledge of the secret. This includes any third party that might be used in the method.
- **Simplicity** This characteristic describes how easy a method can be used. Complex methods are unlikely to be used by parties that do not understand them, because one might not be confident that it can be used safely.
- **Remoteness** Some methods require that all parties are present at the same location while others do not. This principle is expressed by remoteness.

2.2 Contract signing protocols

When two parties that want to compare a secret or the result of a comparison method are at the same location it is possible to show this to each other more or less simultaneously. If the two parties are not at the same location and communicate remotely trough, say, a computer network, this is more difficult. This is because when at the same location you can see the actions of the other party and immediately respond to that. Also you are still in control of you results, you can quickly retreat your result if the other party is not cooperating. However when a message is sent remotely you lose control of it the moment you sent it. As a result, the security characteristic is more vulnerable. In general in the last stage of executing a method both parties have some that they need to compare. If party A sends this value to party B, the latter will know the result, but party A will not know the result until party B also sends a message. The problem is that it is impossible to guarantee that both parties will indeed send the message they promised to send. This may result in one party knowing the result leaving the other in the dark.

A way to reduce this risk is to release the result gradually, say bit by bit where party A and party B take turns. Such an approach is used in contract signing protocols [1] and can be applied to secret comparison methods as well. The key idea is that for both parties it should be equally hard to compute the result using the bits they have already have received. In this case a party can only be one bit ahead of the other so the advantage is small if a large enough number of bits is used.

2.3 Comparing information with and without reference

To distinguish two fundamentally different preconditions in comparing information, Teepe states his definition of "comparing information without leaking it"[1] as:

- *Pieter Noordhuis, E-mail: pcnoordhuis@gmail.com.*
- *Harry de Boer, E-mail: h.de.boer.15@student.rug.nl.*

[1] Is called 'comparing' in further references.

> Two players want to test whether their respective secrets are the same, but they do not want the other player to learn the secret in case the secrets do not match.

This definition does not state which secret is to be compared and how this is decided. It could be that the parties want to compare whether they paid the same amount of money for their new car. This would require both players to first agree on what they want to compare (the amount of money paid for the car), before comparing the value of this secret property. This is called comparing "with reference".

The other possibility is that one party takes a secret (for example: "Alice married Bob") and the other party checks whether he[2] also knows this specific secret. In this case, the parties do not need to agree on the subject of comparison. Both parties posses a set of secrets from which they can choose secrets to compare with other parties. This is called comparing "without reference".

2.4 One-way collision-free hash function

Some methods that are discussed in this paper, rely on one-way collision-free hash functions. Such a function will be denoted as $H(\cdot)$, where multiple arguments represent a concatenation of the arguments. We assume that the following properties hold for this function (the first property satisfies the requirement of being collision-free):

- It is hard to compute two inputs $X, Y, X \neq Y$ where $H(X) = H(Y)$
- For inputs $X, Y, X \neq Y$, it is hard to compute $H(Y)$ if $H(X)$ and the difference between X and Y are given

The first property says that if some third party would intercept the hash computed by this function, it is near to impossible to come up with an input X' that fulfills $H(X) = H(X')$.

The second property says that with a non-empty difference between X and Y, a full recomputation is needed to compute $H(Y)$ if $H(X)$ is given.

2.5 Interactive proofs

Proving the correctness of information comparison methods is simplified by modeling them as a proof system. An NP[3] proof system can be seen as two deterministic machines where one machine acts as the prover and one as the verifier. Formally, the prover is an exponential time Turing machine and the verifier a polynomial time Turing machine. Both read a common input tape and communicate one-way from prover to checker through another tape. The general idea is that it is hard to find an answer but easy to verify the correctness.

Interactive proof systems [4] are a generalization of NP proof systems. The idea of interactive proofs comes from the suggestion that a proof can be more effectively verified if the prover and verifier communicate. For example students probably understand a proof faster when a teacher explains it in front of a classroom where they can ask questions instead of reading the proof in a book. Formally it consists of two Interactive Turing Machines (ITM). Each ITM has a work tape, a tape from which random number can be read, a read-only communication tape, a write-only communication tape and an input tape. A pair of these ITM's share the input tape and the write-only communication tape is the read-only communication tape for the other.

The interactive proof system (IPS) has a pair of ITM's as described above where the prover has infinite power, the verifier is polynomial time and they satisfy the following properties:

- **Completeness:** for an input in the language of the IPS the verifier will accept the proof of he prover with a probability of at least $1 - \frac{1}{n^k}$ for each k and a large enough number of turns n.
- **Soundness:** if the input is not in the language is the IPS the prover cannot, even if it does not follow the protocol, let the verifier accept the input except with probability $\frac{1}{n^k}$ for each k and large enough n.

[2]For every following "he", "him" or "his", read "he/she", "him/her", "his/her".

[3]Non-deterministic polynomial

If a proof system exists for a previously mentioned secret comparison method, the resolution characteristic is implied by the completeness property. The security characteristic is implied by the soundness property.

2.6 Zero knowledge proofs

In 1985, Goldwasser, Micali and Rackoff [4] introduced zero knowledge proof systems. These are interactive proof systems with an extra condition in addition to the ones mentioned in the previous section, being:

- **Zero-knowledge:** the verifier does not get any additional knowledge from the prover than whether the input is true, even if the verifier cheats.

This means for example: if the prover can convince the verifier that a graph has a cycle in it, he can do so without showing this cycle. The verifier *only* gets to know there is one, not where it is or how it is computed. The zero knowledge property corresponds to the leakage characteristic in secret comparison methods. A good overview of zero knowledge is presented in Goldreich [3].

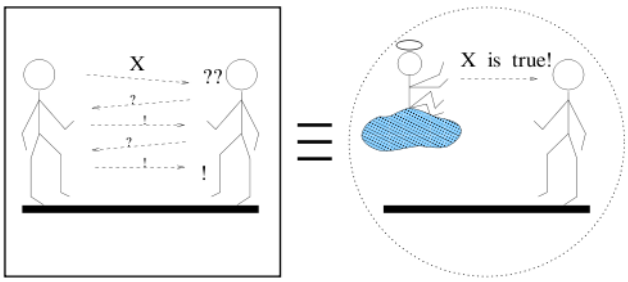


Fig. 1. Zero knowledge, taken from Goldreich [3]

While zero knowledge is an important property of proof systems, we do not give proofs that the methods given in this paper posses this property.

2.7 Correctness logics for comparison methods

In this section we will shortly mention logics that can be used to analyze proof systems. These logics can be of great use to analyze comparison methods for basic flaws.

Firstly there is BAN logic (named after its inventors Burrows, Abadi and Needham) is a logic meant for analyzing authentication methods. It is used to identify whether a method indeed does what it should and on which assumptions the method relies.

GNY logic (named after Gong, Needham and Yahalom) builds upon BAN logic and can handle a greater variety of protocols. An useful feature of GNY for secret comparison methods is that one can define knowledge preconditions. This is important because these should not have been changed after execution of the protocol except for participants knowing if they have the same secrets.

3 Overview of methods

In this section we present current solutions to the problem of secret information sharing. We categorize the methods into categories that indicate their complexity. All methods are taken from [2] unless stated otherwise.

Some of the methods mentioned above use a labeling for candidates. Another way to create labels for candidates is to use a hash function $H(\cdot)$. This is convenient when there is a set with a large number or unknown amount of candidates.

3.1 Simple methods

What we call "simple methods" are methods that can be performed using objects that can be found in common households. That is, they can be used without the use of some technical device or complex mathematics. In a sense these methods could be performed on a deserted island. These may not have real value in computer science but are instructional to get the general feeling for the problem at hand.

- **Trusted party:** Say there is a person both parties are willing to trust. This can be the case when the trustee had no interest in the secrets, then both can tell their secret to the trusted person who can next announce whether the secrets are the same.

- **Cups:** This method assumes there is a pool of candidates for the secret. For each candidate a label is created which is attached to a container (i.e. a cup). Each party now puts a note in each container without the other party seeing what is on the note. The note says "yes" if this candidate is their secret and "no" if it is not. The labels are now removed and the containers are shuffled. If shuffling is done properly the parties do not know which container belonged to which label. Now the containers are opened and if one container contains two notes saying "yes" the secrets are the same.

- **Deck of cards:** Assuming an alphabet of n elements, secrets can be compared using $2n$ cards, n red and n black. Only the color of the cards matter. The cards are put face-down on a table in two stacks, one with the red cards and one with the black. For each letter x_i in the alphabet a pair of red-black cards is drawn. One of the parties picks a red and a black card and puts them face to face horizontally with the black card on top. He takes the cards behind his back so the other party can not see them and flips them if x_i is in his secret. The other party takes the cards and flips them behind his back if x_i is in his secret. The cards are now put on a result stack. When all cards are drawn the result stack is shuffled in such a way that the colors do not show (i.e. using riffle-shuffling). Red cards facing upwards in the result stack indicate that the secrets are not the same since that pair was not flipped by both parties.

- **Envelopes:** Suppose two secrets x and y are represented by a binary number of n bits. Let k be a positive integer, which should be chosen large enough since it defines the probability of false positives 2^{-k}, which should be small. Both parties select $2n$ random numbers between 0 and $2^k - 1$. Each number is put in identical envelopes and each party puts the envelopes in two rows of n envelopes. Both parties now compute a sum $mod\ 2^k$ using their secret and their envelopes. From each column i the top one is added to the sum if $x_i = 0$ (or $y_i = 0$) and the bottom one if that position in their secret is one. The resulting sum for the example in Table 1 is $S_x = (e_1 + e_2 + e_9 + e_4 + e_{11} + e_{12}) \bmod k$

Table 1. Selection of envelopes, boldface indicates selection.

secret	0	0	1	0	1	1
row 1	$\mathbf{e_1}$	$\mathbf{e_2}$	e_3	$\mathbf{e_4}$	e_5	e_6
row 2	e_7	e_8	$\mathbf{e_9}$	e_{10}	$\mathbf{e_{11}}$	$\mathbf{e_{12}}$

Party A now leaves while the other selects the envelopes from party A corresponding to his own secret and puts the other envelopes on a pile. Party A returns and verifies that indeed half of the envelopes are chosen. The pile is now burned and party B calculates the sum as above and adds it to his own sum. Roles are now reversed. When both are finished they write their resulting sum on a piece or paper and lay them side by side. If the results are the same they have the same secret.

3.2 Trusted-party replacements

When mentioning the trusted party above we implicitly meant this party to be human. There are ways to replace this human with some other entity as shown below. A possible secret in some method is referred to as a *candidate*.

- **Computer program:** The trusted party as previously discussed can be replaced by a computer program. This procedure works as follows:

  - Party A enters his secret while party B is not looking.
  - The program clears the screen (even when the secret is displayed with dots or stars, the other party could infer something from the length of the input of the other party) and party B enters his secret while party A is not looking.
  - The screen is again cleared and the program displays whether there was a match or not.

  This method requires both parties to trust the computer programmer that made this program. The program could be made such that it stores all input or returns invalid response.

- **Special purpose device** Another possibility is replacing the human trusted party by a device. One of the properties of such a device could be that it is incapable of storing input. Another property could be that when input is stored in memory and someone tampers with the device, it clears its memory. When using such a device, the manufacturer is the one to be trusted. A special purpose device that - in contrary to a computer program - simply cannot store input, may be easier to trust.

- **Password** It is imaginable that one has a computer at hand but does not want to do new programming or wants to rely on third party tools to match a secret. If so, he can use the tools at hand in almost every operating system. The protocol for matching works as follows (on UNIX, similarly on any other operating system):

  - Party A issues the `passwd` command.
  - Party A is asked for his current password and enters it.
  - Party A is asked for his new password and enters his secret.
  - The program asks for a password confirmation and party B enters his secret. Depending on the output of the program, party B knows whether there was a match or not.
  - If there was a match, the password of party A has changed. The password remains the same otherwise. Party A can also check if there was a match by simply checking if his password was changed.

- **Telephone message** When the secret to compare is part of a limited set of candidates, the parties can agree on assigning a telephone number to each candidate. Party A dials the number corresponding to his candidate and leaves a message for party B. Party B then dials the number corresponding to his candidate and asks whether someone left him a message. When so, he knows that there is a match. To know for sure that party B told the truth when he said there was a match, party A can dial his number again and ask whether party B called to collect the message.

- **Airline reservation** This is a variation of the previous method and it assumes the secret is the name of a natural person. Party A calls to make a reservation for certain flight, that is known to both parties, under the name of his candidate. Next person B calls and tries to cancel the reservation for the name of his candidate. If this succeeds both secrets are the same.

3.3 Cryptographic methods

The methods presented below require some form of calculation. In principle the calculations can be performed by hand so no computer or calculator should be required, but in practice some are only feasible when using a computer. The advantage of executing these methods using a computer is that they easily can be applied between remote locations.

- **Digital envelopes** This is a digital version of the envelopes method stated earlier. Party A sends two envelopes to party B who can choose to receive only the first or the second. This is achieved by using one-out-of-two oblivious transfer [1], which is a method to let the receiving party choose one of two messages but not both. Another property of this transfer method is that the sender (party A) does not get to know which message was chosen to be received. This is equivalent to choosing and destroying the envelopes in the method using physical enveloped described above. In the end both parties are able to compute the sums in the same way as the non-digital method.

- **Random permutation:** This method assumes there is a finite set of candidates. Also there is need for a third party but this person does not get to know the secret. The following steps are taken to execute this method. Each candidate in the set is assigned a unique number. Party A and B agree on a prime number p larger than the highest numbered candidate. They also agree on two random numbers a and b smaller than p. These numbers are not known to the third party. They now calculate $(ax+b)$ mod p or $(ay+b)$ mod p where x and y are the numbers for the first and the second party respectively. The numbers are given to the third party who announces whether the results are the same. If they are the same, the secrets are also the same. Whether the numbers are the same or not, the third party does not know which candidates are talked about since a and b are unknown.

- **Random rotation:** Suppose there is a set of twenty candidates numbered 0 to 19. A third party picks a number k between 0 and 19 and gives this number to party A. Party A calculates $a = (x+k)$ mod 20 where x is the number of the candidate for A. Party A forwards this number to party B who calculates $b = (a-y)$ mod 20 where Y in the candidate for B. Party B gives this number back to the third party who announces whether this number is the same as k, which the third party initially chose. Of course this method works for any finite set of candidates.

- **Hash functions:** Hash functions can be used when two parties want to compare their secrets. Both apply the same hash function to their secret and they compare the result. When the hashes are equal, the secrets are the same. When they are not equal, the parties only saw the hash of each others secret, which is hard to reverse.

- **Padded hash functions (without reference):** Teepe [5] proposes a method which uses padded *one-way collision-free hash functions* to let one party prove possession of a secret to another party. The hash function Teepe uses satisfies the properties stated in 2.4.

  This method provides a way to prove possession of multiple secrets in one go but we restrict the example below to proving possession of one secret.

  We denote the set of secrets party A owns as KB_a (or the knowledge base of party A). Any secret in KB_a is denoted as I_a (or information block of party A).

  In the scope of comparison without reference, one must acknowledge that there is a finite set of secrets each party owns. The sets of participating parties do not need to be the same in size, nor in content. This means when party A sends the hash of I_a to party B, the latter needs to compare this hash to the hash of every secret in KB_b. If a match cannot be found, the protocol can be aborted with both parties knowing that they do not share that specific secret.

  If a third party C would happen to see the hash that party A sends to party B and also happens to own the secret that party A wants to prove to party B, he can conclude by a lookup among his own secrets that party A knows the same secret he knows. Clearly, this is undesirable because party A intended to only prove his possession to party B. To prevent party C from inferring any information from the message passing between party A and B, a nonce N that only party A and party B know can be given as an extra argument of the hash function. By the described properties of one-way collision-free hash functions, we know that party C cannot infer anything from $H(I_a,N)$ if he has some $I_c = I_a$ and does not know N.

  Another way of preventing other parties to eavesdrop on the messages sent between party A and B is by means of a secure communications channel. This could for example be realized by using public/private key-authentication. This would render using a nonce word redundant.

  The key part in this method is that the hashes have to be recomputed each time a message is sent. This prevents a party from pretending he knows the secret, because the original secret is required to compile the hashes.

  The variables A and B denote the unique identity of parties A and B respectively. These public identities are known before execution of the method. If party A wants to prove possession of a secret to party B, and wants to test the knowledge of party B of this secret, the method is executed as follows:

  1. Party A chooses a secret I_a for which he wants to prove his possession
  2. Party A generates a random challenge C_a
  3. Party A sends $h_1 = H(I_a,N)$ and the challenge C_a to party B
  4. Party B finds every $I_b \in KB_b$ for which $H(I_b,N) = h_1$
  5. Party B sends *halt* of he has nothing to prove/verify or a random challenge C_b if he has.
  6. Party A sends the message $h_{2a} = H(I_a,N,A,C_b)$
  7. Party B verifies whether h_{2a} (the received message) is the same as $H(I_b,N,A,C_b)$. If it is, party B can conclude the secrets are the same. Then, he sends $h_{2b} = H(I_b,N,B,C_a)$.
  8. Party A can now verify that h_{2b} is the same as $H(I_a,N,B,C_a)$ to conclude that the secrets are the same and the method is terminated

4 Discussion

For every method to work in practice, participants to the comparison process should agree on how to represent the secret. It is imaginable that parties use different spelling or synonyms to depict the same secret. For a human trusted party, it would be clear that both parties mean the same. Any non-human trusted party will see different entries as a mismatch. Therefore, both parties should agree on a way to unambiguously describe their secret, so a mismatch *really is* a mismatch. This is no issue when there is a candidate set of secrets involved, where each candidate can be given a unique number.

Some methods do allow a slight margin of error. That is, even while the secrets are not the same, the result of the method can be positive with a very small probability. For example, the use of a deck of cards allows different entries to come out as a match when the different secrets are anagrams. Numerical incidence ($5+3$ equals $4+4$) can also lead to false positives. This is possible with the (digital) envelopes method.

In this section we first discuss general methods of cheating followed by a discussion of each method from the cryptographic section.

Table 2. Evaluation of methods using their characteristics.

method	resolution	leakage	security	privacy	simplicity
Digital envelopes	+	++	-	+	++
Random permutation	++	++	++	-	+
Random rotation	++	++	+	-	++
Hash functions	++	-	- -	-	-
Padded hash functions	++	++	-	++	–

4.1 Cheating

When a protocol is followed faithfully both parties should find out the result. However, if one party tries to cheat the results can be different. Cheating can take several forms.

1. One of the parties does not have the secret and tries to find out what the secret of the other party is.

2. One party may want to find out whether they have the same secret, but does not want the other party to find out. This can be achieved when a party aborts the method before completing it, but after acquiring the result he was looking for.

3. An involved third party may not be as reliable as the participating parties might think. The third party might decide to sell the results he received to the highest bidder and return the participating parties false result.

4. When a method is executed it might be that a message is intercepted by an external party and used to steal the secret or convince some party that the external party has the secret while this is not the case. This is especially a danger when the method is applied remotely.

There is another possibility where one party may be falsely convinced that the secrets are not the same. In this case both parties do have the secret, but one party does not want the other party to find out whether they both have the secret but pretends he does. It is hard to prevent this scenario since one cannot make a distinction between whether one party just does not have the secret or is not willing to share. However in most cases it is not very harmful as the cheating party does not find out any information at all (considered the method does not leak information).

The advantage for a cheater trying to use the second form can be reduced using contract signing protocols as discussed earlier. In a similar way eavesdropping by an external party can be prevented by using public key encryption.

4.2 Discussion of individual methods

As characteristics may vary when executing a method remotely instead of locally, we chose to evaluate each method as if it were performed by two computers on a network. Some characteristics can be improved when incorporating public key encryption or a contract signing protocol. Below we will not take these extra additions in account and use the "vanilla" version of a method.

- **Digital envelopes:** When followed faithfully, this method will indeed result in both parties knowing the result. However, this method does not guarantee a valid result will be obtained. When the random numbers are badly chosen (by coincidence), a false positive is generated if different numbers sum up to the same value. The chance of this happening will be small if the numbers are chosen randomly from a domain that is large enough. Another difficulty that is present in many methods is that both parties should get to know the result simultaneously. This problem results in the second cheating method mentioned above being possible. On the positive side, when executing this protocol there is no additional information is gained (leakage) because when the secret is not known, all that can be observed is a composition of some random integers. This also has advantages for the privacy property since the secret itself is never compromised. On the other hand, an external eavesdropper might find out that both parties have the same secret if these numbers are intercepted. This method is relatively easy to understand for a computing science student (simplicity), or even a person not from computing science since it has a real world analogy (real envelopes).

- **Random permutation:** This method uses a third party which means that it this method is suspect to cheating by a third party as described above. Using a third party has the advantage that both parties do not need to tell each other their results simultaneously as these are send by the third party. The disadvantage of course is that the third party finds out whether both parties have the same secret, but the secret itself remains safe. The method is not too hard to understand but it requires some mathematical knowledge to understand why this way of modifying the numbers works to secure the secret from the third party.

- **Random rotation:** When all parties act faithfully this method will result in resolution. The parties that want to compare do not get any information on their respective secrets other than that the secret is in the set of candidates. Only the third party can get to know the difference between the numbers assigned to the two secrets. An advantage is that this method is easy to understand.

- **Hash function:** Using the hash function method the problem is again that both sides need to send each other their hashes simultaneously or one party might cheat. Also there is information leaked since the hash of the secret can be used to convince someone you have the secret when in fact you only have the hash of a secret. If party A starts sending party B can just start return the same hash to convince party A it has the same hash so this method is far from secure. To make things worse this can also be done by an eavesdropper. This method is quite easy to explain but it makes use of hash functions which makes t more complicated.

- **Padded hash functions:** This version improves some of the flaws in the plain hash function method. The leakage problem is solved by padding the input with a challenge. There even is a proof in GNY logic that shows the method will indeed not leak information. This makes it impossible to return the same hash as was received, which enhances the security property. The security problem is not completely solved however. It is still possible for one party to stop communicating after that party found out the result leaving the other in the dark. Because hashes are recalculated each time the method is run a third party cannot gain any information about the data being sent or use data from a previous session to fake knowledge of some secret. The complexity if this method is quite great since to be able to fully understand it in addition to knowledge of hash functions it also requires that variables used for padding are understood. Another advantage of this method is that it is designed to be run remotely. It even has it's own measures to prevent eavesdropping by using a nonce. It is therefor useful even without an encrypted connection.

Table 3. Evaluation of methods using their characteristics with addition of public key encryption and a contract signing protocol.

method	resolution	leakage	security	privacy	simplicity
Digital envelopes	+	++	++	++	- -
Random permutation	++	++	+	-	- -
Random rotation	++	++	+	+	- -
Hash functions	++	-	+	-	- -
Padded hash functions	++	++	++	++	- -

5 Conclusion

This paper presents a theoretically backed way for comparing different methods for comparing information without leaking it. When in the situation where such a method can be helpful when choosing a particular method.

Table 2 gives a high level overview of how each method scored in our discussion. If both parties are willing we suggest to use the padded hash functions method proposed by Teepe. As this method requires both parties to be willing to faithfully participate, this poses a point of security. If some party A receives a hash that tells him the party B proved his possession of a specific secret, he can halt the protocol at that point. This results in party B not knowing whether party A also has this piece of secret information, as he did not complete the challenge.

Therefore, we think it would be of great value if a variant of this method that gradually releases information would be developed. This would let both parties be gradually more convinced of the other knowing that specific secret. It would reduce the impact of problem stated above. If willingness might be an issue and there is a trusted third party available the random permutation method might be a good alternative.

Table 3 shows a level overview of what is to be expected when public key encryption and contract signing protocols are incorporated. In this overview most methods perform better and the padded hash functions method scores are almost perfect. Of course using these additions have impact on the simplicity but we believe it is still feasible to use them.

Acknowledgements

The authors wish to thank Gerard Renardel, Luc Vlaming and Feitze de Vries for reviewing this paper.

References

[1] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[2] R. Fagin, M. Naor, and P. Winkler. Comparing information without leaking it. *Commun. ACM*, 39(5):77–85, 1996.

[3] O. Goldreich. Zero-knowledge twenty years after its invention. In *Electronic Colloquium on Computational Complexity (http://www.eccc.uni-trier.de/eccc/), Report*, volume 63, 2002.

[4] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, New York, NY, USA, 1985. ACM.

[5] W. Teepe. Proving Possession of Arbitrary Secrets While not Giving them Away: New Protocols and a Proof in GNY Logic. *Synthese*, 149(2):409–443, 2006.

A Comparison of Hatching Techniques

Willem Bouma, Erik de Jong

Abstract— The generations of images with the appearance of hatching drawings is a classic field of research in non-photorealistic rendering (NPR). Hatching is an artistic technique used to convey the shape of an object through shading effects created with closely spaced parallel lines. In this work, we draw a component between the different approaches to the computer generation of such imagery. We point out their key features and discuss their advantages and disadvantages. We give an overview of how the different methods work. We then compare particular criteria such as rendering performance, interaction possibilities and aesthetic properties of the result images.

Index Terms— Non-photorealistic rendering, Hatching, Line rendering, Line shading.

1 Introduction

Increasingly more research is being done in the field of *non-photorealistic rendering* (NPR). NPR is about trying to render an image or object using artistic styles such as painting, drawing, technical illustrations and animated cartoons. Each of these styles has its own particular way of obtaining a desired effects.

One of these is *hatching*. Hatching is a artistic style that uses parallel lines to create tonal or shading effects. These effects can be used to convey properties of an object such as material or shape.

A lot of research has already been done on the subject of generating hatching images using computers. With every work having different goals. Some opt for real-time rendering performance, others want to be able to tweak as many options as possible. Every one of these efforts comes with different results which are interesting to compare.

In this paper we are going to compare three of these suggested methods. We will compare them based on certain criteria. Since each work opted for different solutions we compare the results to see how these solutions compare to each other on the different aspects of their algorithms.

We will first summarize the different methods described in the papers. We give a small summary about how the algorithms work. Also pointing out the key features that are brought up in the respective paper. Because we are going to use those advantages in our comparison.

Having discussed the different methods, we define the criteria that we are going to compare. We also discuss the different goals proposed by the researchers of the described methods, these might differ from the actual features.

Having defined the comparison criteria, we compare the different methods with each other.

After the results we will conclude which of the methods works best in certain situations.

2 Hatching Techniques

There are three different algorithms that we will be comparing. In this section, we take a look at how the different algorithms work. We also note what, according to the respective paper, should be the key features of the algorithm. These can be features like real-time rendering capabilities or a very high amount of user interaction possibilities.

2.1 High Quality Hatching

Our first algorithm is the one that is described in [4]. It takes triangular meshes as input. The process of turning the input into a line art image is split up in two steps.

The first step is to preprocess the input and compute 3D lines. The second step is the rendering phase where the 3D lines are visualized.

- *Willem Bouma is a CS student at the Rijkuniversiteit Groningen.*
- *Erik de Jong is a CS student at the Rijkuniversiteit Groningen.*

In the preprocessing phase, the 3D lines are computed. Therefore the direction of the lines is established first. A *direction field*, based on the curvature of the surface is then computed.

To avoid unwanted artifacts caused by high levels of detail, a simplification algorithm is applied to the input mesh. Simply put, the mesh gets smoothed before the curvature is computed.

After the preprocessing stage comes the rendering phase. One of the goals the researchers set for themselves, to render the output in high quality images. To get the desired result they set several smaller goals.

The first goal is to have a fast and effective line processing, including *hidden line removal* (HLR). The second one is to have a smooth transition between different tones. The last goal is to output vector-oriented data. This last goal comes in handy when you would want to change certain aspects later on, zooming in and out for example.

The first step is to remove the lines that should not be shown in the output, this is obtained by performing HLR. The HLR described in [4] uses a *z-buffer* based method. This does result in some imprecisions, but these can easily be solved by removing all the strokes on back facing triangles. This way it allows for interactive frame rates.

The shading of the lines is done using a rich NPR centric shading model. This provides a lot of options for the user to tweak. It allows for easy adjusting stroke width, and stroke density.

In addition to creating single hatched images, it is also possible to create multipass images by combining several rendering passes. For example, you could generate *cross-hatching* by rotating the direction field.

An example of what the output could look like is shown in Figure 1. Figure 1(a) shows the result after just one pass. While Figure 1(b) shows the result after three passes, every pass having a rotated direction field.

2.2 Real-Time Hatching

This section covers the algorithm that is describes in [2]. This method is designed to achieve high frame rates and allows for real-time rendering. This is achieved by using textures. This is a common technique to render complex details. These textures contain prerendered hatching strokes. The surface is then rendered by appropriately blending the textures.

Because textures are normally not used to convey shading of an object there are extra steps taken to show the tones of an object, without lighting. So the range of tone values and computes the right tone value using lighting computations. The object is then rendered without light and with the toned textures.

Since the strokes on a texture work only on one level of detail there exists a problem. When the rendered object is scaled, there is the problem that the gaps between strokes become too big, while strokes should be appearing to fill these gaps. For this, a technique similar to the one described in [1] called *mip-maps*. These mip-maps contain the textures for several levels of detail. When zooming in on a rendered object, extra strokes appear to fill the gaps between the other strokes.

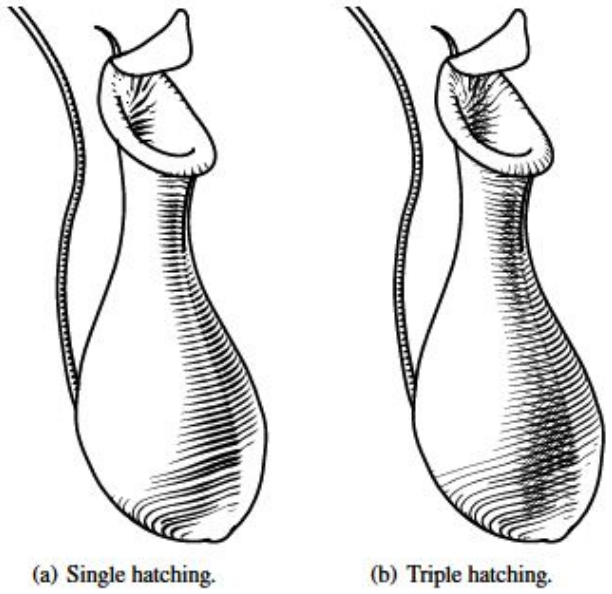


Fig. 1. Example of single pass hatching (left) and three pass hatching (right) using the algorithm from [4]. Image taken from [4].

This set of textures is called a *Tonal Art Map* (TAM). An example of a TAM can be seen in Figure 2.

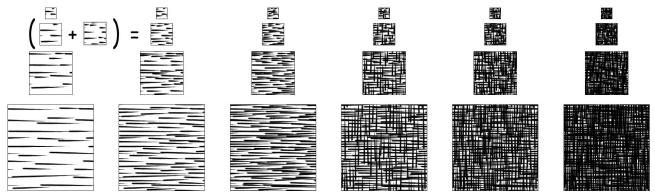


Fig. 2. Example of a Tonal Art Map, the left most strokes appear in all the other textures. Image taken from [2].

TAMs can hold any type of stroke, such as pencil or crayon strokes. The strokes on the textures can be drawn by hand, or generated automatically which is discussed in [2].

For the rendering of the actual output, there is some extra work that needs to be done. The tone of a mesh will not have a smooth transition. To get this, the algorithm blends between two consecutive tone images at each vertex of the mesh. For each gathered vertex the blended textures are combined over the entire face of the mesh.

This already gives a good result for most cases. But it is still possible to get some grey toned strokes. These strokes are unwanted for pen-and-ink strokes. Thresholding removes these unwanted strokes, the algorithm uses a transfer function which gets the wanted result. Although this also has a disadvantage, carefully anti aliased edges might look jaggy.

Examples of some results can be seen in Figure 3.

2.3 Line-Art Rendering of 3D-Models

The method described in [3] is a method that relies on the user for input. The user can set a number of reference strokes for the algorithm to work with. These strokes are used for the curvature and shading of the object.

The algorithm consists of three separate phases.

During the first phase of the algorithm the mesh data is preprocessed. The mesh data has to be mostly manifold, but small non-manifold artifacts can be handled.

For all vertices in the mesh, the principle curvature directions are computed. The stroke lines of the line art rendering will follow these curvature directions. During this phase, the user will interactively control this geometric detail looking at the mesh for curvature analysis.

These geometric attributes are set for the vertices, so that interpolated versions of these values can be used on the triangles. The mesh is then rendered to an *enhanced frame buffer* where different kinds of pixel values are stored, such as local shading (used later to determine the stroke density), normal data (used for contour extraction) and the projected curvature values.

In the second phase, the image stored in the buffer will be segmented, once again aided by the user. The information in the frame buffer will provide some basic information about the contours, but the segmentation is mainly considered to be an artistic process. for each segment, a *fishbone* structure is generated that contains all the necessary information for the actual stroke generation. Different kinds of image processing filters are used to remove turbulences in the direction field.

In the third phase the actual tone mapping is performed. The directional and shading information are translated into the appropriate stroke width, the strokes are spread as uniformly as possible on the surface.

An example of the output is shown in Figure 4.

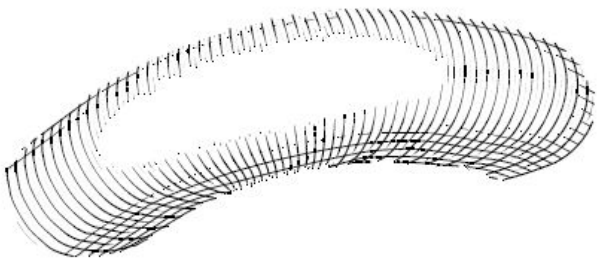


Fig. 4. Example output generated by the method described in [3]. Showing the uniformly spread strokes. Image taken from [3].

3 Comparison Criteria

Since we are going to compare the different algorithms we should define which criteria we are going to compare. Each algorithm has its benefits and drawbacks, these are normally pointed out in the respective papers. The goals that the researchers set themselves at the beginning of their research also influence these strong- and weak points. For example, a very fast algorithm might not allow a lot of user adjustments.

We compare several of the different aspects of the algorithms. But we will first line out the different goals of the algorithms. This will be used in the conclusion to see whether the goals have been met or whether they have been surpassed by an algorithm that did not even intend to excel in it.

The goal described in [4] is to generate a method that produces high-quality renditions. Although these renditions take more processing power to be rendered. The method also allows for a high level of user interactivity. This includes modifying viewing, lighting, model transformation parameters and adjusting different rendering parameters.

Our second method is the one described in [2]. This method clearly focuses on speed. The goal of the algorithm is to deliver real-time

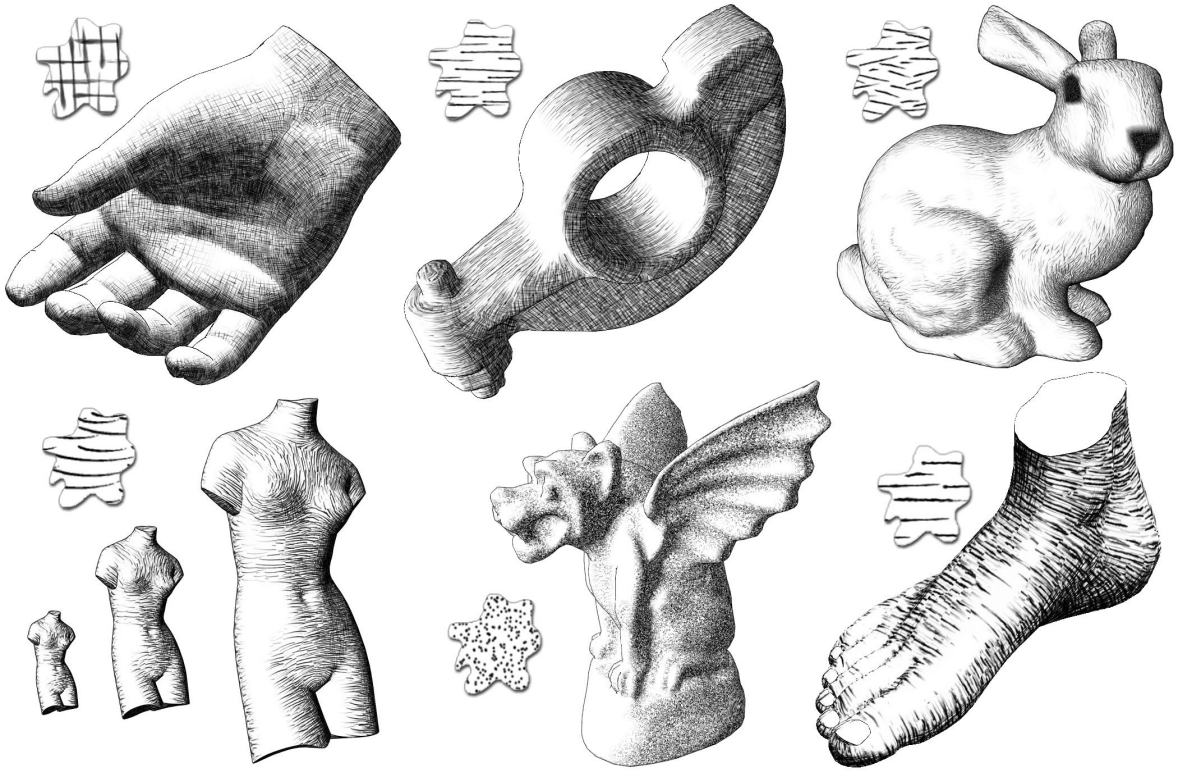


Fig. 3. Six models rendered with different Tonal Art Maps, the inset shows the textures patches. Image taken from [2].

results that can be used in different environments such as games or interactive technical illustrations. Because of this requirement user interactions, such as zooming, rotating, of both the light sources and model, and using different TAMs are also very fast.

The algorithm described in [3] sits a bit in the middle of the other two methods. The user does not get a lot of ways to adjust parameters after the process. However, the method does allow a lot of interaction possibilities in the process. It is possible to set some reference strokes, which will be used by the algorithm. Speed is not a real issue for the researchers. This is the only method that produces images in image space, rather than leaving the input in object space. These images are meant to be used as technical drawings.

As can be seen above, each of the methods has its own particular goal. Some of these overlap, while others do not. Besides these goals, we also compare applicability, usability and the aesthetic properties of the results. Applicability compares how the different algorithms can be used for different tasks. Some of these tasks are mentioned in the respective paper.

Usability is about the ease with which a user can achieve a desired goal. Aesthetic properties can be summed up as saying how nice something looks. This is a very subjective aspect of hatching images. Since not everybody has the same opinion about certain styles.

4 Comparison

We have defined the criteria for the comparison. In the following, we draw a comparison of the different methods according to the defined criteria.

4.1 Rendering Performance

The comparison of rendering performance has one remark. The algorithm in [3] creates strokes in image space, for a fixed perspective projection, while the other two methods allow for interactively altering the view, lighting and object transformations. The method described in [3] is fairly fast otherwise. It uses hardware acceleration for the preprocessing steps. So the resulting image is rendered in a short time.

But it is not really a fair comparison, so we will focus more on comparing the other two methods. Since the more interesting part about the speed aspect is being able to interactively alter the view, lighting and object parameters, for these it is required to redraw the image on the screen over and over.

The algorithm in [4] is slower due to more costly operations like curvature estimation, optimization and generation of strokes in object space. However, it does not make explicit use of available hardware resources. Some optimizations are done during the processing of the data, such as hidden line removal, but these do not have a major impact on the performance of the method.

Our last method, [2], on the other hand makes full use of available hardware. Since this method uses textures it covers larger objects much faster than algorithms that draw their own lines. This is due to the algorithm only needing to fill triangles in the right direction, rather than having to compute curvature estimation, optimization and generating the strokes in object space like the algorithm in [4] needs to.

Results of the method show that interactive frame rates are already achieved with a GeForce 2, with frame rates averaging 28 frames per second. Considering the capabilities of nowadays graphics hardware, one can expect instantaneous renderings from this method.

4.2 User Interaction

The amount of user interaction one would need depends greatly on the task a user would want to fulfill. If all a user would want is an basic image generated out of 3D input there is not a lot interaction needed. Just like that, someone would want more control over the looks which can be obtained by using different methods.

The goal for the algorithm in [4] was to make a program takes gives a lot of freedom to the user on how to adjust parameters of the rendering. Settings such as usage of cross hatching, crossing only once or several times are to the users disposal. An example of this can be seen in Figure 1. The strokes the algorithm generates are obtained from a formula that defines the stroke style. This formula can be adjusted interactively. But whether you want this is another question,

since putting in a formula is not very intuitive. Regardless, the other algorithm do not have this kind of interaction. There is also the ability to adjust stroke density and width. However, it is not possible to how the strokes are rendered other then moving the light source.

The method described in [2] allows the user to easily modify the camera position, light source positions and model transformations. However it is not possible to change parameters such as stroke density and stroke width, other than changing the TAM for the rendering. This is a very inflexible way of working, as it does not allow to quickly adjust the stroke appearance.

Having discussed the other two algorithms we are left with one algorithm. The algorithm in [3] is the only algorithm that has mandatory interaction of the user. This is not a bad thing, but it is also not a very beneficial feature. The user must first set a number of reference strokes for the algorithm. These strokes are used as reference for the interpolation of the hatching strokes done by the algorithm.

As can be seen above, there are many different parameters that can be adjusted, whether this is before, during or after the rendering. Some algorithms have more options than others. Some parameters are not as easy to adjust as others, such as creating your own TAMs or coming up with you own formula for the algorithm from [4].

4.3 Applicability

In this section we will compare the usability of the different methods. Since one method is not likely to be the best in every possible situation we take a look at the types of images that the methods are best used for.

The algorithm described in [3] already describes a useful purpose for the method, technical drawings. The images that are generated with this method are very clear without any areas with a real high density of strokes. Since the user must first set a number of reference strokes, the curvature of the result is always clear. Images generated like this can be used in books and manuals. An example of this can be seen in Figure 5.

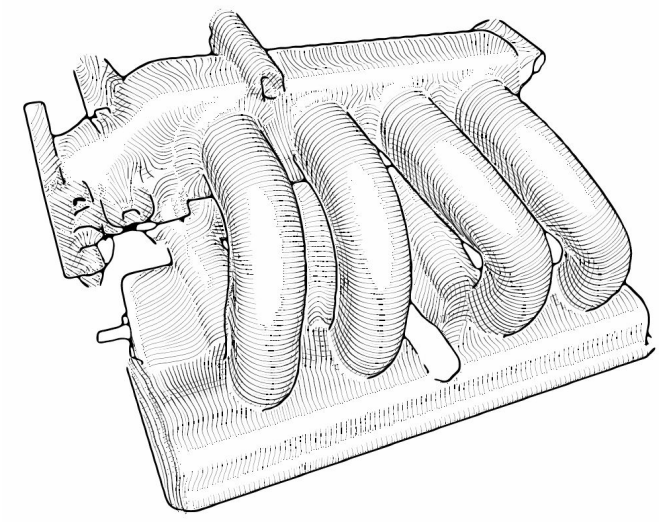
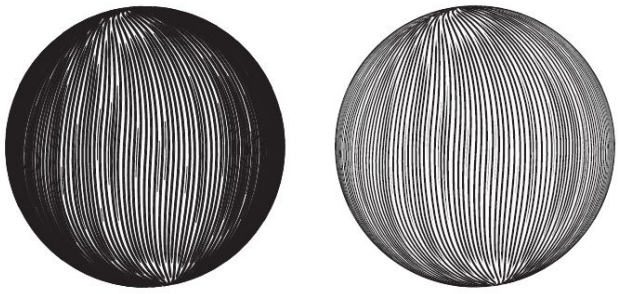


Fig. 5. Technical line render of an engine part. Created using the method described in [3]. Image taken from [3].

The method from [4] also seems to be good for generating images that can be used in books and manuals. However, the user should probably adjust some settings to get a more clear and cleaner image like with [3]. But with the amount of possible user interaction this should not be a problem. This method also has some extra features that can be used to generating images that can be used for printing. One of these is a stroke density correction factor which is demonstrated in Figure 6.



(a) Without a correction factor. (b) With the correction factor.

Fig. 6. (a) shows a sphere drawn with hatching strokes without use of the correction factor. (b) shows the same sphere making use of the correction factor. Image taken from [4].

Unlike the other two algorithms, the algorithm from [2] does not seem to be very effective when it comes to making images that can be used in books, as the resulting images have too much of an artistic look. Therefore they are not very good to convey curvature as clear as the other algorithms are. Because of the high speed of the algorithm it would be much better for it to be used for tasks that require such speed, games for example. But since the images look pretty artistic there are other possible applications for this method, such as rendering images that look like sketches. Although it is not limited to that, since the TAMs could simulate numerous different artistic media, such as pencil, ink or crayon, it can also simulate different artistic styles such as stippling.

4.4 Usability

To compare the usability of the images that are generated by the different methods we want to use a hand drawn as comparison. Figure 7 shows a hand drawn image which makes use of hatching strokes. Because it is hand drawn, the creator had full control over the lighting, shades and curvature of the strokes. Our methods do not have that possibility because they are limited to what the results of the algorithms.

The benefit of using TAMs in [2] is also its drawback in this case. For one, the TAM can not be changed from region to region. It is also not possible to create long strokes, as the length of the stroke is limited by the size of the TAM. Meaning that the entire image would use the same textures. Figure 7 clearly has regions where there are only horizontal strokes, or only vertical strokes. This algorithm would only be able to distinct them based on the light, but this would still not result in our reference image.

The method used in [3] tends to create much cleaner images than Figure 7. This would result in less strokes in certain areas, while some other areas would probably get strokes while there should be none. This could have been avoided if the algorithm let the user set the stroke density and width.

Our last algorithm is the one from [4]. The high level of possible user interaction can be a great help, but this algorithm does not give us the possibility to decide where strokes should appear. Being able to set the stroke style with a formula could work out pretty well. The hand drawn image uses the same sort of strokes throughout the image, but it is still questionable whether the algorithm would pick up the different vertical and horizontal lines.

None of the algorithms can get a result that looks similar to Figure 7. There might be other examples where the algorithms perform better, but this would be different for each algorithm. Therefore these algorithms should not be used to replicate another image.

4.5 Aesthetic Properties

This property is not really comparable. Aesthetic properties are very subjective, it really depends on what someone his or her preference

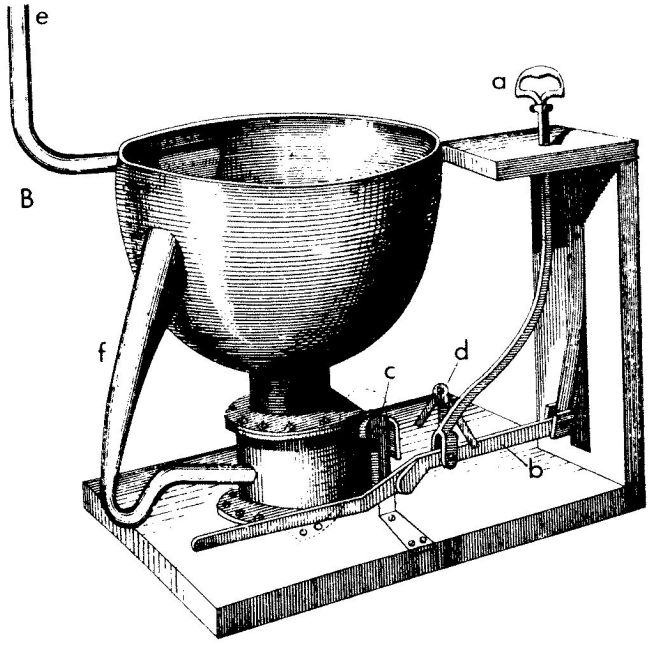


Fig. 7. A hand drawn image that is used for comparison of the aesthetic properties of the different methods. Image taken from [3].

is for certain styles. However, we can discuss some aspects that are specific or common for the resulting rendering of the algorithms.

The algorithm from [4], has a very precise, clear and even look. This is emphasized by the use of evenly spaced and regular strokes. It can create very long strokes as can be seen in Figure 1. Singular strokes are clearly visible and easily distinguishable. The images resemble pen-and-ink illustrations found in medical and botanical works. However, the images may appear unnatural or synthetic. Because of this the images are easily distinguishable from hand drawn illustrations. The renderings convey the curvature of the surface very precise, this is why the images are suitable for the earlier mentioned medical and botanical works.

Similar to this is the algorithm from [3]. This algorithm also creates images that have a very precise and clean look. It also uses very evenly spaced and long strokes, that are very regularily spread over the surface. There are some notable differences, however. This algorithm uses low contrast strokes, this on top of using very slim strokes results in a very regular tone over the entire image. As this results in very light images, without any dark regions, the image has a very flat feel. The reference strokes the user sets, to segment the image, are visible in the resulting rendering. The images generated by this algorithm resemble illustrations you would see in technical works.

Completely different from the previous two algorithms is the one described in [2]. This algorithm uses very short strokes, and depending on the TAM also very dense strokes. Singular strokes are less visible, but they are also less important for this algorithm. It uses a much higher contrast than the other two algorithms, this makes the shading much more expressive, as can be seen in Figure 3. It also has a more hand drawn appearance than the other algorithms, which have a more synthetic appearance. It is also the only algorithm that does not solely use black strokes, but can use any shade between white and black, actually, it could use any color you could put in the TAM. The TAMs also allow for simulating different artistic media, while the other methods can only do hatching. This makes the algorithm suitable for different tasks.

Looking at the results we can see that the algorithms described in [3] and [4] are fairly similar, while the method from [2] gives completely different results. Because this section is about aesthetic properties it is up to oneself to decide what looks nicer.

5 Conclusion

In this work we have compared three different hatching algorithms on a number of criteria. Each of the algorithms has its own criteria where it performs well, or in completely different ways than the other algorithms. The results of these comparisons are interesting, as the results are not as obvious as it sometimes looks.

We listed the different goals the researchers of the other works. We will discuss whether these goals have been met or not. Therefore we will go over the different criteria that have been compared.

The oldest method we compared is the one in [3]. The initial goal of the method was to be able to create images from a 3D-model, which would then be used as technical drawings. The method is not very fast in rendering one image, although it does make some use of available hardware. This is not a problem since the user will most likely spend more time on settings the right reference strokes. What we did see is that the result images are good, as can be concluded from the comparison of applicability. The usability tends to suffer from the mandatory reference strokes. Because the result images are clean and precise the images convey the information they should to be used as technical drawings. However, this method does not allow for transformations of the object, camera or light sources which makes it less usable.

A much newer method that has similar aesthetic properties is the method described in [4]. Rather than focusing on one type of output image, this method was meant to give the user a lot of parameters to adjust. In our comparison we concluded that this method can be used for medical and botanical illustrations, as this method gives very clean and precise results. The user can adjust a lot parameters with this method, but some are not as intuitive as others. Adjusting camera angle, and light sources is easy, but if you want to change the stroke style you need to fill in a formula, which is not very intuitive. Since the method generates images with long and broad strokes, which convey the shape of the object very good, we concluded that the method is suited for generating medical and botanical illustrations. The reason for this is that in these fields it is very important to have a precise description and illustrations of objects. Although some of the features are not as intuitive, the algorithm can be used easily to create high quality images, just as the researchers wanted, which was the other goal for the research.

The third method is the one described in [2]. It was stated clearly that they researchers wanted a method to generate hatching images at interactive frame rates. From our comparison, we can conclude that this is the case. Due to the TAMs used in the algorithm it can make full use of available hardware and render images really fast. The TAMs leave the user free to choose from different artistic styles, although creating a new TAM is not a trivial matter. The result images are very expressive, but are limited to very short strokes. The method allows for easy adjustment of parameters such as camera angle, light source or object transformations. Since the algorithm is very fast it can be used in fields that require fast data representation, such as games. Although, thanks to the TAMs, it can also be used to simulate artistic media like pencil or crayon.

From our comparisons we can conclude that the different goals set by the researchers for their work have been met. Some of the methods have obtained some extra features, that were first not planned. This, of course, is not a bad thing. For all the methods, the user gets enough parameters to adjust to their liking. The rendering performance of the different methods is also very good. Although two of the three methods did not focus on achieving this kind of performance, still ended up with a fast algorithm that would easily achieve interactive frame rates given nowadays technology.

It shows that there are many aspects to the research in the field of hatching. Each of the methods has its own field of specific illustrations and artistic styles. The methods might not be as intuitive for certain aspects, like creating your own TAMs or formulating a new formula, but these methods make up for that with the other parameters that can be adjusted.

In the end there is no universal method that can be used in any situation. To generate an image, that has certain properties requires some research. It is also likely that not all the desired properties can be fulfilled, also because there is still a lot of research being done in the field of NPR. It is possible to point out a preferred method in certain situations, but that does not necessarily has to be one of the methods described in this paper, and a lot of comparing and research has to be done to actually find such a method.

6 Future Work

It would be interesting to do a much broader comparison. Comparing more algorithms would result in some very interesting results regarding the different comparison criteria.

It would also be interesting to see how methods with the same design goals in mind work out when compared to each other. For example, compare other real-time algorithm with the method in [2].

It would be interesting to obtain the implementations of the different algorithms. Having them could really show the differences between the methods. At this point we were only able to compare them using the informations that was provided to us in the papers.

References

[1] A. Klein, W. Li, M. Kazhdan, W. Correa, A. Finkelstein, and T. Funkhouser. Non-photorealistic virtual environments. In *SIGGRAPH*, pages 527–534, 2000.

[2] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *ACM SIGGRAPH*, pages 581–586, 2001.

[3] Christian Rössl and Leif Kobbelt. Line art rendering of 3d-models. In *Pacific Graphics*, pages 87–96, 2000. 10.1109/PCCGA.2000.883890.

[4] Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High quality hatching. In *Computer Graphics Forum*, volume 24, pages 421–430, 2004. 10.1111/j.1467-8659.2004.00773.x.

From Software Requirement to Architecture Design: A comparison of the methods

Marcel Koster, Lazaro Adolf Luhusa, *Student, RuG*

Abstract—In the Software Engineering field there are several methods to build an architecture from a set of requirements. Companies often do not know what method is the best method for their needs. This paper will examine three of these methods and compare them according to an example and according to the expertise needed for applying each method. The outcome will provide a basis on which companies can select the method which suits the company and their needs.

Index Terms—Software Requirements, Architecture, Design, Patterns, Twin Peaks.

1 Introduction

In the field of Software Engineering there are many obstacles and difficulties when building an architecture. One of the most important and often failing requirements is making sure that the actual delivered system satisfies the stakeholders concerns. If this fails, a lot of time and money can be wasted. To ensure that this does not happen, various methods can be used to match the architecture to the requirements. The problem is that most companies do not use any of these methods and the ones that do, often use a wrong method. It is important to use a method that fits best to the project at hand, because wrong methods can cost more time and money than using a suitable method.

Today the widely used methods for building an architecture are the waterfall model [5] and Pattern-Driven partitioning [3]. The waterfall model is an old and no longer satisfiable method. It has some major problems, such as being unable to adapt to changing requirements or stakeholder concerns. Pattern-Driven partitioning on the other hand is better suited to make this match.

There are various methods to compose an architecture from a set of requirements. We will discuss three of these methods. The first method is an adaptation of the Twin Peaks model [1]. The second is Pattern-Driven partitioning. The last is called Goal Oriented design, which is using a repository of Reusable Generic Architectural Drivers (RGAD) [4]. We will describe these methods in chapter 2.

We intend to compare the proposed methods in three ways, which is described in chapter 4. First we compare them according to the amount of Requirements Knowledge and Experience (RKE) needed, which is explained in [2], to assess the expertise needed for each method. The second comparison will show their similarities, differences, advantages and disadvantages. Thirdly we compare the methods by applying them to an example of system requirements, from which we make an architecture. This example is explained in chapter 3. Finally we come up with recommendations on choosing the best suitable method in chapter 5.

We expect that the best suitable method for our example will be Pattern-Driven partitioning, because it provides a complete analysis from requirements to architecture. We also expect that Pattern-Driven partitioning is the method, which needs the least amount of RKE, because it provides clear-cut patterns ready for use.

2 Architectural Design Methods

We will first discuss the methods themselves to give a clear overview of how they work.

2.1 Twin Peaks Model

Requirements and architectures are the core concepts to arrive at the concrete software systems that satisfy stakeholders concerns. The twin peaks model addresses requirements specification and architecture specification concurrently and independently. The requirements and architecture go from the level of general to the level of detailed and both are not prioritised. This can be seen in Fig. 1. The method is purely iterative and starts when the first requirements are provided. There are no clear steps involved in this process. The process is further explained in [1].

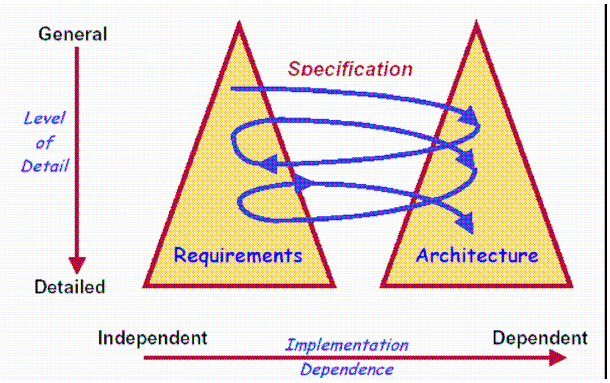


Fig. 1. Twin Peaks model [1].

2.2 Pattern driven architecture partitioning

Pattern driven architectural design is about applying patterns according to certain key drivers derived from the requirements. The architecture takes shape as the patterns are combined to form the system. The pattern driven method is composed of the following steps:

1) Identify the most prominent architectural drivers of the system.
2) Select candidate architecture patterns that address the needs of the architectural drivers.
3) Partition the system by applying a combination of the candidate patterns.
4) Evaluate whether the partitioning satisfies the architectural drivers. This step may include the following activities:
   a) Examine the forces of the pattern to understand any challenges posed by the solution, and determine whether they apply to the current system.
   b) Examine the consequences of the patterns to determine whether the architectural drivers are satisfied.
   c) Examine the consequences of the patterns to determine whether they impose constraints on the system that may require additional patterns, or may render any patterns impractical for this system.
   d) Examine the interaction among the patterns selected to see whether the impact on the quality attributes are compatible.
5) Perform trade-off with respect to the different architectural drivers.

This process can be repeated as additional architectural drivers are identified, or if fundamental attributes of the system change. In subsequent iterations, it is normal to consider the impact of existing patterns on the newly identified drivers before adding new patterns; i.e., begin with step 2.

The results of this approach form a starting point for the architectural design. The next steps vary in different architecting methods, but they usually involve further decomposition, documentation of other views, and eventually architectural evaluation and evolution studies. Fig. 2 shows a state diagram of the process, which is clearly explained in [3].

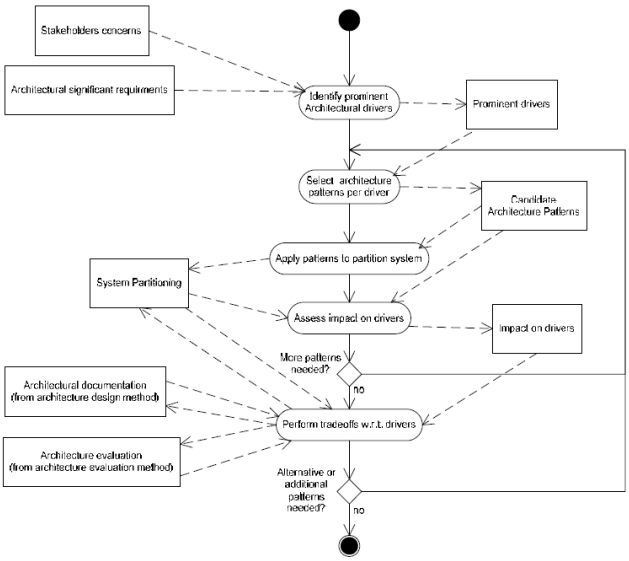


Fig. 2. Pattern driven architecture partitioning [3].

2.3 Goal oriented requirements engineering and compositional architecture development

Goal oriented requirements engineering is all about selecting goals for the system and choosing reusable components that satisfy these goals. First, a goal model is made, which is a model used to identify the mission of the system to be built. These goals identify what the system should do. The main modelling elements that are found in the goal model are goals, soft-goals, tasks and the relationship between them. Goals are the functional requirements of the system. Soft goals are the non-functional requirement of the system. Tasks are used to model the goals. From these goal models the architectural drivers are selected, which will be mapped to generic architectural drivers in a repository. The reusable components will then be selected. This method is composed of the following steps:

- Identify high level set of requirements from which the system has to be built.
- Identify concrete architectural drivers.
- Map the concrete architectural drivers with the generic architectural drivers in the repository.
- Select reusable architectural components.

The steps are clearly visible in Fig. 3. The process itself is explained in [4].

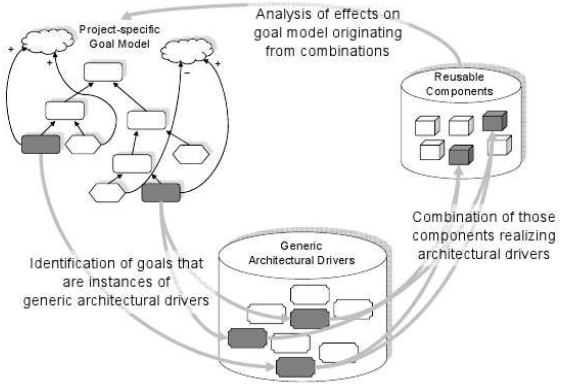


Fig. 3. Goal oriented model [4].

3 APPLICATION EXAMPLE

To compare these methods we will use an example. We will follow the steps provided in each method and create an architecture according to a set of given requirements. We will compare the results later on. In our example we will create an Ebay system, which has the following requirements:

- Online auctions:
  - There are several possible scenarios of how auctions are run, to have flexibility in setting up and running auctions.
- Searching:
  - The system will be able to search active and past auctions.
  - There will be several possible search modes, such as search by general category of merchandise, keyword in item title, by seller, by location of item (e.g., search only auctions in a particular country.)
  - It is expected that this database will grow large over time, and as you can imagine, it is very dynamic. Auctions are starting and finishing all the time.
- User accounts:
  - They allow users to set preferences.
  - They allow buyers to see which auctions they are involved in.
  - They allow sellers to see their active auctions.
  - They allow people to rank satisfaction with sellers and buyers.
  - A person may be both a buyer and a seller.
- Payment:
  - The system handles payment by handing the transaction off to a third-party payment system.
- Security: The system has strong security needs.
  - Personal information must remain safe, and auctions must not be compromised.
  - Transactions (e.g., bids) must be secure.
- Uptime:
  - The system must be up 99.99% of the time.
- Transactions:
  - The system must handle transactions reliably. For example, since networked communication is vulnerable to disruption, partial transactions must be handled or prevented.
- Performance: Performance must be consistent. In particular, an auction must end exactly when it says it will, and bids must be handled promptly and consistently
- Notification System:
  - Seller is notified about each bid on her auctions
  - A buyer is notified if her bid is surpassed
  - A buyer is notified if she has won an auction

3.1 Twin Peaks Model

The first method to be tested is the Twin Peaks model. There are no clear steps involved in this method. We should just start with a set of general requirements and create a basic architecture for that. In the next iteration we will go into more detail. We first consider the following general requirements for the E-bay system: Online auctions, Searching, User accounts, Payment, Security, Uptime, Transactions, Performance and Notification System.

According to these general requirements, we first come up with a basic client-server architecture with multiple clients to allow Online actions. We then add a User database and an Auction database, to support User accounts, Online auctions and Searching. Next is a Payment system for Payment and Transactions. The last is a Notification system. Security, Uptime and Performance is difficult to put in a system and are too general to give any clues as to where they are needed. The composed system is shown below in Fig. 4.

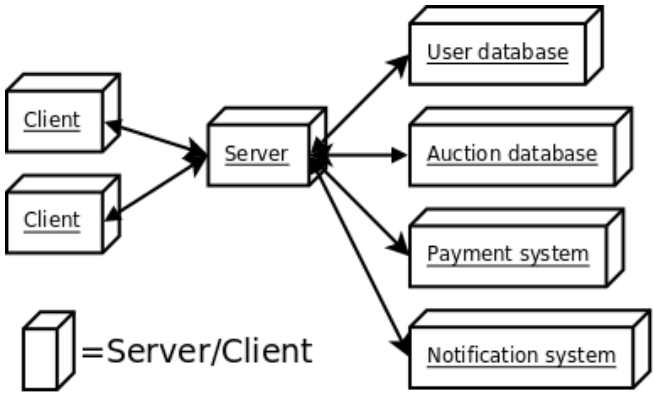


Fig. 4. Initial architecture.

The next step in the Twin Peaks method is going into more detail. We will now consider the whole set of requirements as stated earlier as well as fictional feedback from the users of the system. According to our fictional users the system is not fast enough when multiple users are accessing the server. When considering the whole set of the requirements, some changes have been made to the initial architecture. The Databases have been replaced by Repositories, which contain multiple databases. This is to ensure that the Databases can be dynamic and that their growth is supported. The Auction Repository has been split up into Past Auction and Active Auction Repositories. This is done to ensure that the system can search past and active auctions without losing too much Performance. A 3rd party now provides the Payment system. There can now be multiple Servers, to ensure Performance and Uptime. Security is not met in this diagram, as it is still too unclear as to how this should be added to the system. The elaborated system is shown below in Fig. 5.

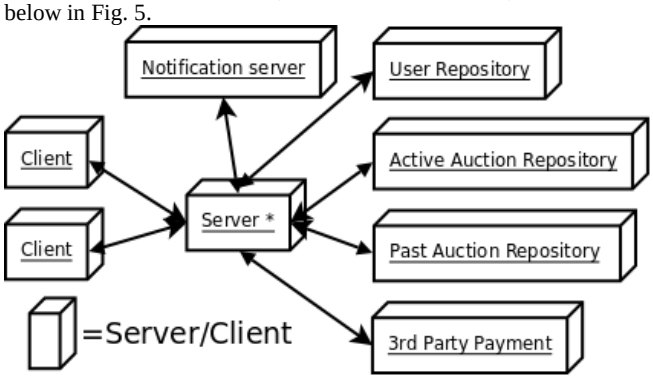


Fig. 5. Elaborated architecture.

3.2 Pattern driven partitioning

The second method to be tested is Pattern driven architecture partitioning. We will follow the steps provided in this method. We will first identify the most prominent architectural drivers of the system. According to the requirements we have come up with the following key drivers: Security, Availability (depends on Reliability and Scalability), Usability and Reliability (contains Fault tolerance). Security because of the security demands on the system, Availability because of the uptime, Usability because of the user accounts and Reliability because of reliable transaction handling.

The next step is to select the candidate patterns that address the needs of the architectural drivers. The following patterns are selected: Broker (Good for: Security; Neutral for: Fault tolerance), MVC (Good for: Usability), Microkernel (Good for: Reliability), Layer (Good for: Security) and Shared Repository (Good for: Fault tolerance, Scalability; Neutral for: Security). These patterns combine to the following system, shown in Fig. 6:

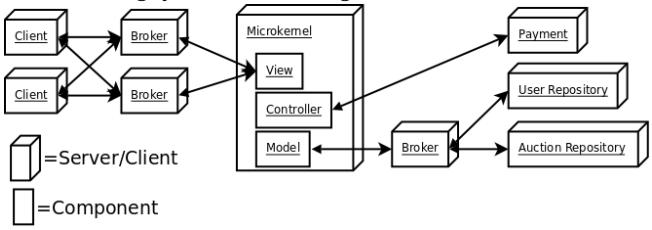


Fig. 6. Architecture with patterns.

In this system there are multiple Clients connecting to multiple Brokers. These Brokers connect to the Microkernel and especially to the View layer. The Controller layer handles connections to the Payment system and to the Model. The Model connects to the backend Broker, which can access the User and Auction Repository.

3.3 Goal oriented

The third method to test is Goal oriented design. We will again follow the steps provided in this method. We will first identify a high level set of requirements for which we will again use the next list: Online auctions, Searching, User accounts, Payment, Security, Uptime, Transactions, Performance and Notification System.

We will now identify the concrete architectural drivers. We have come up with the following drivers, which of course are the same as in the Pattern Driven method: Security, Availability, Usability and Reliability.

From this we have made the following Goal model, shown in Fig. 7. In this model we have left out the Tasks, which would only clutter up the diagram and don't add any meaning in our example. The rounded squares are the Goals and the ovals are the Softgoals.

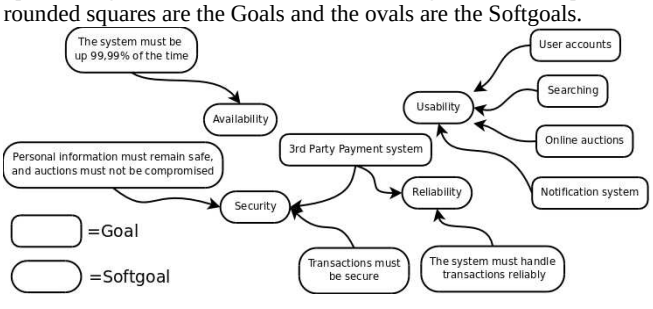


Fig. 7. Goal model.

The next step is to map the concrete architectural drivers with the generic architectural drivers in the repository. Of course we do not have a repository with generic architectural drivers, so we decided to use documented patterns as our repository, which are listed with their architectural drivers in the POSA book [7]. If we then select the reusable architectural components, we come up with the same

system as the Pattern Driven method. This will be explained in the comparison.

4 METHOD COMPARISON

We will compare the methods with three different approaches: by looking at their advantages and disadvantages, by comparing the amount of requirement knowledge and experience needed for each method and by looking at the findings of our own example.

4.1 Requirement Knowledge and Experience(RKE) comparison

Requirement engineering (RE) and software architecture (SA) are the most important parts for the delivery of software systems. Software architects with requirement knowledge and experience (RKE) will perform better than those without RKE [2].

We have observed that there is a gap between requirement and architecture in the Twin Peak Model, as specified in [1] and there is no systematic approach to choose patterns in the model and in this case we can say that the Twin Peak Model can be achieved with software architects with RKE and will bring difficulties to Non-RKE architects.

We have also observed that the pattern driven architecture has a systematic approach of choosing the patterns based on the architectural drivers. This method favours both the RKE and Non-RKE software architects although the RKE will always perform better.

The Goal oriented architecture favours the RKE software architects when the software organization builds their first software. There will be no known architectural drivers to compare at this step and even no software components to reuse. But when the organization is an experienced organization and has documented the architectural drivers (Generic architectural drivers) and components, then both the Non-RKE and RKE software architects perform better.

4.2 Advantages and disadvantages comparison

Each of the methods has it's own advantages and disadvantages, which we will compare here.

4.2.1 Twin peaks model

The Twin Peaks model addresses the three management concerns as identified by [6]:

- IKIWISI (I will know it when I see it). The design and implementation starts earlier than usual and explicitly allows the user to provide feedback (Prototyping)
- COTS (Commercial off the shelf) software. Reuse of components at an earlier stage of requirements specification
- Rapid Change, adapting to rapid changes which allows quick development and competitive marketing.

The Twin Peaks model has some advantages. The model leads to software that is adaptable, because of intertwining the requirements and the architecture. The model evaluates changing project risks and funding, because it is very flexible. There is a speedy delivery of the product to the market and the feedbacks are considered in the further development. The model supports reuse of existing software components and reduces the costs of development by allowing reuse of components early.

The model has some disadvantages as well. First of all there is a gap between requirements and architecture, which provides difficulties when trying to make them fully coherent (working together) during early stages of development. This is because the level of detail for requirements is so general. The model does not satisfy specific quality attributes, as they are not identified. The model produces an unstable architecture due to changes in the requirements, which were not prioritised.

4.2.2 Pattern driven

The Pattern driven design has some major advantages. First of all Architecture patterns show singular power in linking functional and non-functional consequences of an architectural approach together. The solutions have been verified through extensive experience, which leads to understanding of their impact on non-functional aspects of systems. An architect can use the information found in architecture patterns to help make decisions about how to design the system. Architecture patterns can also help architects identify when an architectural approach might introduce conflicting approaches to different quality attributes. This additional information may cause the architect to carefully consider the usage of this pattern before it becomes too late. Patterns have rich relationships among them. These relationships may include the following: A pattern may influence the use of other patterns, a pattern may specialize the use of another, two or more patterns may be alternatives, etc. This helps when an architect is trying to reason about the consequences of a combination of patterns upon the quality attributes. It also helps the architect to find alternative patterns as solutions to the same problem. Patterns can be used as part of the natural flow of architectural design. Patterns tend to emerge naturally during the course of architecture, and their use can be easily documented without disrupting the architectural design process. Architects can use them almost naturally within the context of almost any architecture process they use.

One of the disadvantages of the Pattern driven method is that Architecture patterns may increase or decrease the understandability of a design or implementation. They can decrease understandability by adding indirection or increasing the amount of code. Another point is that Architecture patterns do not lead to direct code reuse. One of the most prominent disadvantages is that Architecture patterns are complex in nature, but look deceptively simple. The advantage that Architecture patterns are validated by experience and discussion is a disadvantage as well, because they have not been validated in any other way.

4.2.3 Goal oriented

The Goal oriented method supports reuse of the architectural knowledge and hence the method helps novice architects to arrive at an architectural design. The method also provides a good way to build the system that satisfies the stakeholders concerns, because of its way to take the feedback from the architectural design back to the requirements engineering. Furthermore, it's a systematic approach that allows traceability and reduces the cost for development by reusing components.

Apart from these advantages, there are some disadvantages as well. First the method may cause delay to the architectural design when generic architectural drivers do not match with the reusable components. Limitation to the reusable architectural components may cause the system not to be built, especially if the architect is a novice in architecting. The method cannot be applied to the first software project of the company since generic architectural drivers must derived from previous IT projects within the same organization.

4.3 Comparison according to example

We have observed that all three methods have requirements as inputs and architecture as outputs. We have seen that the architecture obtained by the Twin Peak Model will not satisfy the quality attributes of the system since changes will be implemented without caring about the drivers. The architecture, which we have made, does not pay attention to Security and Reliability. The system will work, but it will not satisfy all the requirements. Above that, it was difficult to come up with an architecture because there was no easy way of finding a suitable documented architecture type.

The Pattern driven architecture does satisfy the quality attributes of the system and each time the architecture is iterated, the candidate architectural drivers are taken in consideration. However it may be difficult to provide an early prototype for this system in order to receive feedback from the stakeholders, which is much easier with the Twin Peaks model. It was not very difficult to come up with an architecture with this method, because of the documented patterns.

Similarly the Goal oriented architecture satisfies the quality attributes of the system (or key drivers), because it does not differ a lot from the Pattern driven method. The only problem with Goal oriented design was that we did not have a database with generic drivers. The advantage is that you already have the components and do not need a lot of implementation.

5 CONCLUSION AND FUTURE WORK

Systems are built to satisfy software requirements and its implication is to satisfy the stakeholders' missions and goals. A Software architecture is derived to present the views in which the software will be built.

We have chosen three methods for deriving the requirements to architecture. We firstly mentioned the methods, Twin Peak model, Pattern driven design and Goal oriented design. We explained the steps provided to go from requirements to architecture with each method. We found it necessary to understand the three methods, before we created an example. We then applied all three methods to the example and observed how each method affected the requirements and architecture.

We have compared the methods based on specific variables, which were RKE, observations, advantages and disadvantages. We found that Pattern driven partitioning was the best method of the three since it performs better than the other methods. Not only did the method favour both experienced software architects and non-experienced software architects, it is also a very good method for software organizations because of the amount of documented patterns. The Twin Peaks model can be a good method to follow, but you will need experienced software architects to ensure that the architecture is correct and can be implemented in time. The Goal oriented method can also work very well, but only if there is a rich database of documented components ready for reuse.

Our study is a base for a decision making process, for choosing the best suitable method one can use, to get from software requirements to a matching architecture. We propose more research to be done on these three methods and other methods so as to strengthen our study. We have done theoretical research, from which we have made our conclusions, but we recommend industrial practice in the three methods, in the same software project, for further research. We propose more research and stimulation to ensure correct and widely spread usage of software design methods by companies.

- *Marcel Koster is with RuG, E-Mail: M.Koster.5@student.rug.nl.*
- *Lazaro Adolf Luhusa is with RuG, E-Mail: L.A.Luhusa@student.rug.nl.*

REFERENCES

[1] Bashar Nuseibeh. Weaving Together Requirements and Architectures. 2001.

[2] Remo Ferrari and Nazim H. Madhavji. The Impact of Requirements Knowledge and Experience on Software Architecting: An Empirical Study. 2007.

[3] Neil Harrison and Paris Avgeriou. Pattern-Driven Architectural Partitioning: Balancing Functional and Non-functional Requirements. 2007.

[4] Sebastian Herold, Andreas Metzger, Andreas Rausch, and Heiko Stallbaum. Towards Bridging the Gap between Goal-Oriented Requirements Engineering and Compositional Architecture Development. 2007.

[5] Dr. Winston W. Royce. Managing the Development of Large Software Systems. 1970.

[6] Barry Boehm ("Requirements that Handle IKIWISI, COTS, and Rapid Change," Computer, July 2000, pp. 99-102)

[7] F. Buschmann, R. Meunier, H. Rohnert, P.Sommerlad, M. Stal John Wiley and Sons Ltd, Chichester. Pattern-Oriented Software Architecture: A System of Patterns , UK 1996 . ISBN 0-471-95869-7

Interactive displays in our homes, now or in the future

M. Gjalt Bearda, Luc Vlaming

Abstract—Nowadays there are quite some devices with touch displays. We looked at how multitouch displays can be used at home. We compared the most common and readily available techniques available today in how they could be used at home. We looked at various properties of these touch displays, namely orientation, text input, borders and objects, games, and long term usage. We looked at various use cases, comparing the properties important for that use case. The use cases considered are playing games, a work table, designing/drawing, and a digital cookbook. We conclude that the Cintiq scores best overall. For the games use case however, the DiamondTouch is preferred. With the digital cookbook the DViT layer and the Cintiq have equal preference.

1 Introduction

Nowadays there are small devices which can be controlled via touch input; popular examples are mobile phones like the iPhone (Apple), the iPod Touch (Apple), the Touch HD (HTC), G1 (HTC) and the Blackberry Storm (RIM). Also systems like ATMs with touch displays are being deployed. These displays are larger, but mainly used by companies.

Touch displays can provide input in substantial different ways than the traditional mouse and keyboard setup. In a household, these devices could be used for example for playing advanced board games on a table with several people or in the kitchen providing cookbook recipes. Other things that could be done with large touch displays at home, are watching interactive tv, drawing, things like ordering photos or commenting papers.

Because touch displays already appear in handhelds and laptops, it could be argued that, in some time, these devices could appear in household devices too. There are a lot of places where a touch display could appear in a household situation. For this reason we did a comparison between several commercially available touch display products on their usability in a household setting.

In this paper we look at how multitouch displays can be used at home. We compare the most common and commercially available techniques available today. The products compared in this paper are the DiamondTouch (MERL) [DL01], DViT [Tec03], FTIR [Han05] and Cintiq (Wacom) [Wac07]. We chose these devices because there are easy to buy, and in this way could be placed in homes right now. Also to our knowledge these devices had the most extensive research.

We look at various properties of touch displays, comparing properties important for household use of touch displays. The specific properties we research are tilting/placement, text input, borders and objects, games, and long term usage. We chose this properties, since they are reported most in our references as problems with tabletops in the settings the research took place in. Also, some of these properties (tilting, borders and objects, games, and long term usage) could prevent touch displays from being used at households.

In the related work we specify the devices. Then we specifiy for each property what research has been done and what the results are. In the discussion we look into each property for all devices and specify the problems and positive sides of it. After the discussion, we specify the results for a few use cases which seem relevant in household usage.

2 Related work

There is some related work in this area, it is noteworthy that there has not been a lot of research into long term usage of tabletops except for one case study. This, however, is quite understandable since the field is quite new.

- *M. Gjalt Bearda is with Rijksuniversiteit Groningen, E-mail: mgbearda@gmail.com.*
- *Luc Vlaming is with Rijksuniversiteit Groningen, Email: asperientje@gmail.com.*

2.1 The devices

For each device we look at a few points which influence our comparison:

- How is touch registered
- How is the screen displayed
- Can pressure be sensed
- Can users be uniquely identified
- Can objects be placed on the screen

We look at these properties of the devices so we can refer to them later on. Because we do not have the resources to test the devices we want to compare, we use the specifications provided by the manufacturers.

2.1.1 DiamondTouch

The DiamondTouch is a capacitive system, which allows the unique identification of users using the tabletop. This is achieved by sending a small current via the user to create a loop with the table when the user touches the table. Because the current flowing through each user uses a different wavelength, each user can be uniquely identified. This, however, creates the necessity that the users need to be connected to the tabletop in order for touches to be registered (they need to sit on a special fitted chair, or need to keep a cable in their hand).

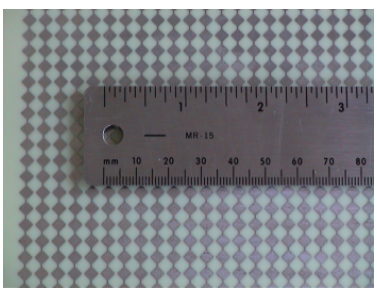


Figure 1: The antennas used in a DiamondTouch. From [DL01]

Touch is registered when a user touches the table, in this way creating a loop. The table works with antennas arranged in rows and columns. In Figure 1 the column arranged antennas are shown. When a user touches the surface, it connects several row and column antennas together. In this way the position of the user can be estimated.

Pressure can be sensed in the way that the harder the user presses, the bigger the contact area is, thus touching several antennas and registering a bigger touch surface.

This technique does not include a display in the tabletop, so the display needs to be projected on the tabletop. This also means that,

when a user has their hand above the table, a "shadow" is dropped, and below the hand no screen is visible. However, in [RME+06] their observation is that this is rarely a problem. They even state people do not even notice a projector is used. Others even take advantage of the top projection, since the content is still visible for example on their hand.

The DiamondTouch allows arbitrarily objects to be placed on the tabletop, since these devices do not create a loop, and are not registered as touches. Since there is also a protective layer over the antennas spilling coffee, for example, is not a problem.

2.1.2 DViT

DViT stands for "Digital Vision Touch". The system uses four cameras placed in the corners of a screen. DViT layers can be used to make an ordinarily screen multitouch. Unfortunately the maximum number of concurrent touches on the screen is not specified in the white paper, but we estimate that around two users should be able to work together on one screen at most.

Touch is registered by the cameras, which see the fingers in their view. Because the cameras are calibrated, only two cameras are needed for each finger to be triangulated.

Because every normal screen can be fitted with a DViT layer, a normal screen provides the display. This way the display does not require any projection hardware, and has no requirements for space to the display.

This device does not allow arbitrarily objects to be placed on the screen, since these will possibly obstruct the view of the cameras, and also be registered as touches.

2.1.3 FTIR

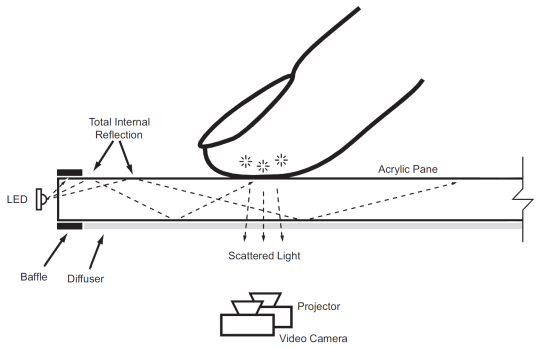


Figure 2: The FTIR technique. From [Han05]

FTIR stands for "Frustrated Total Internal Reflection". For this technique a thick sheet of acrylic is used, wherein infrared light is sent. Because of total internal reflection, the light stays in the sheet. When a user touches the sheet, this effect does not happen anymore at the position where the user touches the sheet. This results in infrared light being sent out of the sheet on the place of the finger, which is registered by a camera. This results in blobs in the camera-image, which can then be interpreted as touches. Also pressure can be sensed, since if a user presses harder, more light will be sent out and therefore the blobs will grow.

In order to get a display on the acrylic layer, the acrylic needs to be fitted with projection paper, on which a projector can project its display. Since this technology allows the display to be projected from the rear as well from the front, most FTIR displays are rear projected, since this prevents occlusion problems. This results in either a deep display or a separate display and projector and also limits the placement of the displays.

Placement of objects results in touches, since placing an object on the screen results in light being sent out to the camera, resulting in touch detection by the display. However, when using a history image of the view (generating a "scenery" image) these kind of objects can be filtered out by software.

2.1.4 Cintiq

The Cintiq is a stylus based touch technology. It uses a pen to sense where the display is touched (unfortunately how this is not published in the white paper), and sends the touch position together with pressure information and the pen identification number to the computer. This results in a multitouch display, where multitouch is created by the styluses. This way the number of concurrent touches is limited by the number of styluses. The sensor board is fitted together with a normal display, so no projection is needed.

Because a stylus is used and the pen identification number of the pen is sensed, it can be sensed which pen touches the surface.

Placing objects on the surface should not create a problem.

2.2 Tilting/placement

(a) Slightly tilted. From [WPR+07]

(b) As a coffee table. From [RME+06]

Figure 3: Tilting and placement of a DiamondTouch.

Tilting is a quite important subject, since researchers found that people like to tilt their workspace. For example in [WPR+07], there was reported that AB (the study participant) liked to tilt the workspace as much as possible. The orientation used is shown in Figure 3 (a). In [MBM08], also the study participants wanted to tilt their workspaces in an arbitrarily angle. The tilting of the workspace happened when a single user uses the tabletop as their workspace.

When collaboration occurs between multiple people using one tabletop, the preferences for orientation and tilting of that table change [RME+06]. When collaborating using one large tabletop, people like to have their own personal space in order to still feel comfortable working with each other. When people know each other, this constraint is less pressing, but in various cultures this issue can be bigger [MHM+08].

Orientation is important in a home usage setting since multiple people will be using a tabletop when it would be placed for example on the table in the living room. There has been done some research in this field by [RME+06], showing that problems could arise when someone is using the tabletop and someone else wants to start using the device too, and "steals" someone else's objects. Their research shows this can be fixed by using a bigger table. They observed 80cm diagonal is too small, their minimum observed size was 107cm diagonal.

Also, the placement of a tabletop is important since not all tabletops have integrated displays. For these products, this creates the problem of placement of the projector, since there should be enough distance between the projector and the tabletop to create a large enough projected display.

2.3 Text input

Batch text input on tabletops is a problem, since there is no regular keyboard, with it tactile feedback. There are no complete solutions available yet to input the same amount of text with equal comfort as with a keyboard. There are several partial solutions for this;

however, most of them do not approach the performance or accuracy achieved with a normal keyboard. A solution available, used in [RME+06, WPR+07], is using an on-screen keyboard (like the windows on-screen keyboard). Another solution is using a stylus and write on the screen, together with handwriting recognition software. The last solution is to use speech recognition, but only for a subset of commands, as used in [TGSF06].

Some people however seem to be able to adapt to using an on-screen keyboard on a touch tabletop. This is shown in [WPR+07], where the study participant AB writes even longer emails with his tabletop on-screen keyboard than with a normal keyboard.

People also like handwriting recognition together with a stylus [MBM08] to compose word-processor documents. The study participant in [RME+06] felt the process of hand-writing a document allowed her to reflect on it carefully, while the automatic conversion of the ink to text then created a product the study participant could share with others.

Text input can be important in a home usage setting, when desktops computers are replaced completely in the home. However, as shown in [WPR+07], this is not necessarily a problem. It should be noted though, that AB (their study participant), said himself people would be crazy when using an on-screen keyboard for everyday use.

2.4 Borders and objects

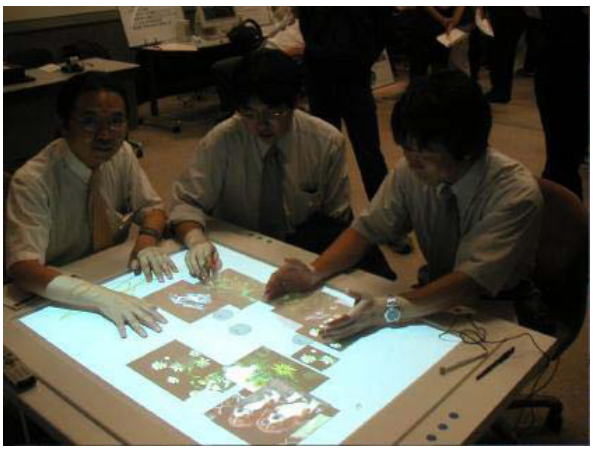


Figure 4: A DiamondTouch with a large border. From [RME+06].

Quite a lot of researchers found out that people have problems with accidently touching the table. For example, people place their elbows on the table (see Figure 4), providing ambiguous touches on the tabletop [RME+06]. In order to minimize these accidental touches, the tabletop can be fitted with an insensible border.

Also ambiguous touches can be created due to the fact that people accidently touch the interface without the intention to create interaction on the table, for example, when someone is pointing something out to someone else. As far as we know, people have not yet found a good solution for this.

Other problems that arise are due to the fact that people want to use the tabletop (when used as an addition/replacement of the workspace) as a table instead of a "fragile" screen [WPR+07, MBM08]. For example, in [MBM08] is observed people feel not like placing objects on the screen. People have problems with this, and say they want to put the tabletop aside, so is does not interrupt their normal workflow.

Borders on a tabletop and the ability to place objects on a tabletop without touches being registered are important aspects on the usage of a tabletop in regular, everyday use of a tabletop.

2.5 Games

At home people like to play games, whether they are collaborative games or not. Tables are used for playing board games, however, doing games with a table as interaction board is not yet common. In [TGSF06] it is observed that people who are playing a board game still are engaged and still play a role even though the game itself is turn based. It is also observed that using a tabletop adds a new dimensions. For example, when speech recognition together with multitouch is used in strategy games. This adds to the experience of the user.

This means that tabletops, when placed on a table in, for example, the living room, can be used to play these kind of games.

The aspect of games should be looked at in our research too, since this can be a big aspect in the acceptance level of a tabletop in a household.

2.6 Long term usage

To our knowledge, only one long term usage field study has been conducted to date. This research [WPR+07] was done at MERL, and was a field study of one person using a DiamondTouch [DL01] for an extended period of 13 months for everyday work, as a replacement of his normal desktop. There most intriguing results were that the study participant did not have problems with arm fatigue, which to the researchers knowledge is because the field of view / fatigue trade-off is heavily skewed towards the field of view.

Another surprising fact is that, after extended usage, the study participant did no longer need specific gestures in order to select small UI components.

Because we want to incorporate tabletops into people their homes, looking at longterm usage for devices is relevant. Since we only know about one longterm study participant, the comparisons here may be skewed towards that study participant unfortunately.

3 Discussion

We have done a comparison between the products for each property important for household usage.

3.1 Orientation

For comparing the different tabletops with respect to the orientation there are a few differences due to how the display is created (with a projector or not). If one would want to wall-mount these devices, the Cintiq and DViT have no problem with this. However, the DiamondTouch will need a projector projecting the display on it, possibly with an angle to avoid to much occlusion. The FTIR based technique has some more problems, since generally these devices are rear-projected. This generates quite a thick device to be mounted on a wall, requiring a thick wall.

With respect to tilting the devices (when used as a workspace), again the projector-based devices have some problems, since either the projectors need to tilt too, or they need to support tilted projection surfaces. Most projectors nowadays support projecting on tilted surfaces, so this will not be a big problem most times.

When used by more than one user, the interface will be oriented most times either completely horizontally or completely vertically.

When used horizontally, this results in a scenario with a table interface. For usage as a table interface, the DiamondTouch, and Cintiq displays do not have big disadvantages (the DiamondTouch is designed to be used as a table). However, the DViT layered interface will have problems when more than two persons interact with the device, because the triangulation of the fingers gets problems with more than 2 concurrent touches. This can be solved by taking turns. The FTIR based display however requires a camera beneath the display, so the table starts to be more thick, which could be a problem. Another problem both the FTIR and DViT based displays have, is that they can not uniquely identify which user is touching. This can be a problem when users are for example playing board games.

When used vertically, this results in a wall mounted interface. In this usage scenario, all the considerations with the horizontal display apply, with a few additional remarks. The DiamondTouch needs the users to hold something, or to stand on a sheet in order to create the electrical circuit to identify users. Also it needs the projector to project on the screen without the users occluding it too much, possibly by

projecting from the top with an angle. The FTIR and DViT and Cintiq based techniques work do not have any new problems here.

3.2 Text input

Text input could be important, but only if we would remove the normal ways of entering text into word-processors. This point should only be taken into consideration if the other standard ways of word-processing would be removed from the households. For word processing there are several global possibilities: an on-screen keyboard, using a stylus or using a wireless keyboard.

Using an on-screen keyboard is not recommended by most researchers. The study participant in the long-term usage even says that no reasonable person would use an on-screen keyboard [WPR+07] (he considers himself then to be a non-reasonable user).

Since the main way of writing before digital word processing appeared has been with a pencil, using handwriting recognition with a stylus can be handy. Research reveals [RME+06] that study participants even like the stylus, because they feel that the process of handwriting a document allows them to be able to reflect on it carefully, while the automatic conversion of the ink to text then created a product the study participant could share with others. However, handwriting recognition software have not yet developed enough to recognize people's handwriting without adaptation. Therefore, people need to adapt their handwriting. Since quite some people already use handwriting recognition on PDAs nowadays (which also needs adaptation of their handwriting), this does not seem to be a huge problem. Handwriting with styluses could be used on all the devices, so there is no real preference for any of the devices here.

The third option, to use a wireless keyboard, does not add to the comparison of the devices, but is a good way to input loads of text.

3.3 Borders and objects

Placing objects on the tabletops is an important issue, since study participants report on the tabletop as being in the way of their normal workflow if they can not put objects on the tabletop. In this perspective, especially the DiamondTouch is a good candidate to use, since this product allows arbitrarily objects to be placed on the tabletop without ever interfering with the interface. However, also FTIR could be well adapted to this usage scenario. The DViT is not really resistant to placing objects on its screen, since it uses cameras in the corners of the display to recognize touches. When a big object is placed, apart from it being registered as a touch, it could occlude quite some part of the display. The Cintiq will not have big problem when objects are placed on it, since only the styluses are recognized providing input to the Cintiq.

3.4 Games

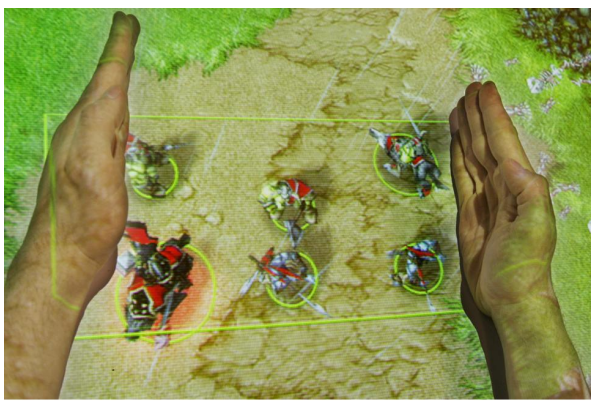


Figure 5: Example gesture used to select a region. Taken from [TGSF06]

When tabletops are used with games, people tend to be more varying in the gestures used. The support for rich gestures (for example selecting a region with the sides of your hands; see Figure 5), can be important when playing games, because the more intuitive the gestures can be, the easier the adoption of the tabletop will be, and the more frequently people will use it. The Cintiq and DViT based devices do not allow for this kind of gestures. To play games which allow a rich set of gestures, this can be a problem.

When playing a multiplayer game, several aspects play a role. Uniquely identifying the user is important, because in this way only the user itself can play when the user has the turn (when playing a turn based board game). Here, the DiamondTouch and Cintiq are preferred. However it is not specified how much styluses the Cintiq can support, so we are not sure about this.

Another aspect is the number of concurrent touches an interface supports. Here, the DiamondTouch, Cintiq and FTIR based techniques are preferred, because they support more concurrent users.

3.5 Long term usage

Since only one paper has been published about long term usage of any tabletop yet, we can only say something about long term usage with respect to this device. The things we can say in respect to this research, is that people like to tilt their tables as much as possible [WPR+07, MBM08] until things started to slide across the surface. This is possible with all the techniques researched, so this is not a problem.

4 Results

For tabletops, there are a few cases in which a tabletop could be used in household usage, weighted by the criteria given in the discussion:

- (Board) games
- Work table
- Design/drawing table
- Digital cookbook in the kitchen

For each of these cases we give a suggestion on which product would be the best, together with the measures used to come up with this suggestion. For long term usage, we can unfortunately for none of the cases really give any preferences for any of the products since we could only report on one study participant, so this property will be left out in the comparisons.

4.1 Board games

With traditional board games, and most games played by more than 2 players, the display is preferably used in a horizontal setting. Text input is almost never the case in games, so this is ignored. The orientation, borders and games properties will be considered in this case.

Properties	Borders and objects	Orientation	Games
DiamondTouch	+	+	+
DViT layer	-	-	-
FTIR technique	+-	+-	+-
Cintiq	+	+	+-

For the borders and placing objects on the displays, the DiamondTouch and Cintiq win, since they allow to place objects on the screen without any problems. The FTIR technique can do this, but with adjustments, and the DViT layer can not handle these kind of things very well. However it should be noted that these are all minor differences, and do not play a big role for the comparison.

For the orientation, again the DViT layer and FTIR technique have problems with objects on the table. Also the DViT layer has problems with more than 2 touches.

The DiamondTouch and Cintiq allow to uniquely identify the users. The DiamondTouch and FTIR technique allow for two-hand selection-like gestures (explained earlier). So in overall, for the game property, the DiamondTouch wins.

We conclude that overall the DiamondTouch is the preferred product in this case.

4.2 Work table

With a work table, the display could be oriented in all kinds of situations. Text input should be possible. The games property is not important, since the display will not be used as a game.

Properties	Orientation	Borders and objects	Text input
DiamondTouch	-	+	+-
DViT layer	+	-	+-
FTIR technique	+-	+-	+-
Cintiq	+	+	+

For the orientation, the DiamondTouch and FTIR technique have a problem, since possibly they would need to project the display at an angle (when top-projection is used). When rear projection is used, FTIR has a smaller problem, but still tilting could be a problem. Multiple touches are no problem here, since most times only one person would be interacting with the touch display.

When working, it is handy to put stuff on the table. The only device which would have a real problem with this if the DViT layer. The FTIR technique could have problems with this, but they should be minor.

For text input, the only real winner is the Cintiq, since this device uses a stylus. All the other devices should be able to work more or less with styluses too, but there are not designed for this purpose.

We conclude that overall the Cintiq is the preferred products in this case.

4.3 Design/drawing table

With a design or drawing table, the display could be oriented in all kinds of situations. Text input could be necessary, for example when making notes on a design. The games property is not important, since the display will not be used as a game.

Properties	Orientation	Borders and objects	Text input
DiamondTouch	-	+	+-
DViT layer	+	-	+-
FTIR technique	+-	+-	+-
Cintiq	+	+	+

For the orientation, the DiamondTouch and FTIR technique have a problem, since possibly they would need to project the display at an angle (when top-projection is used). When rear projection is used, FTIR has a smaller problem, but still tilting could be a problem. Multiple touches are no problem here, since most times only one person would be drawing. If more people would be drawing at a time, it could be we would want to uniquely identify the users, but this would rarely happen, so this is not be a big issue.

When drawing or doing design, it is handy to put stuff on the table. However it is not necessary, and could be avoided. The only device which would have a real problem with this if the DViT layer. The FTIR technique could have problems with this, but they should be minor. Since placing objects on the tabletop can be avoided, we do not see this as a major problem.

For text input, the only real winner is the Cintiq, since this device uses a stylus. All the other devices should be able to work more or less with styluses too, but there are not designed for this purpose.

We conclude that overall the Cintiq is the preferred products in this case.

4.4 Digital cookbook

With a digital cookbook, the display will probably oriented vertically. Text input should not be necessary in the kitchen (use as a cookbook), so this is ignored. The borders and games are not important, since it is not a game, and since we do not lean on the display, borders are neither a problem.

Properties	Orientation
DiamondTouch	-
DViT layer	+
FTIR technique	+-
Cintiq	+

For the orientation, the DiamondTouch has a problem, since the projector would need to project at an angle. Since in most kitchens there is not a lot of space above the cupboards where we could for example hang our projector, this gives problems. For the FTIR technique, there are some problems too, since the camera needs to be behind the display. For this, the display gets quite thick, probably even too thick to fit in the cupboard (if we would for example put the projector there too).

We conclude that overall the Cintiq or DViT layer are the preferred products in this case.

5 Conclusion

We have made an comparison between several devices, the Cintiq, the DiamondTouch, FTIR based techniques, and DViT layered displays. The Cintiq is the winner in most of the use cases, except for the games case. When playing games the DiamondTouch wins. Also for the cookbook use case, the DViT layer is a winner.

References

[DL01] Paul Dietz and Darren Leigh, *Diamondtouch: a multi-user touch technology*, UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology (New York, NY, USA), ACM, 2001, pp. 219–226.

[Han05] Jefferson Y. Han, *Low-cost multi-touch sensing through frustrated total internal reflection*, UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology (New York, NY, USA), ACM, 2005, pp. 115–118.

[MBM08] Meredith Ringel Morris, A.J. Bernheim Brush, and Brian R. Meyers, *A field study of knowledge workers use of interactive horizontal displays*, IEEE International Workshop on Horizontal Interactive Human-Computer Systems (September 2008).

[MHM+08] Paul Marshall, Eva Hornecker, Richard Morris, Nick Sheep Dalton, and Yvonne Rogers, *When the fingers do the talking: A study of group participation with varying constraints to a tabletop interface*, IEEE International Workshop on Horizontal Interactive Human-Computer Systems (September 2008).

[RME+06] Kathy Ryall, Meredith Ringel Morris, Katherine Everitt, Clifton Forlines, and Chia Shen, *Experiences with and observations of direct-touch tabletops*, IEEE International Workshop on Horizontal Interactive Human-Computer Systems (January 2006).

[Tec03] Smart Technologies, *Digital vision touch technology*, Tech. report, Smart Technologies, 2003.

[TGSF06] Edward Tse, Saul Greenberg, Chia Shen, and Clifton Forlines, *Multimodal multiplayer tabletop gaming*, International Workshop on Pervasive Gaming Applications (PerGames) (May 2006).

[Wac07] Wacom, *Tech paper - interactive pen display*, Tech. report, Wacom, 2007.

[WPR+07] Daniel Wigdor, Gerald Penn, Kathy Ryall, Alan Esenther, and Chia Shen, *Living with a tabletop: Analysis and observations of long term office use of a multi-touch table*, IEEE International Workshop on Horizontal Interactive Human-computer Systems (December 2007).

Primitives of Lock-Free Algorithms

Nikolaus Manojlovic

Abstract— In computer science, non blocking synchronization ensures that threads competing for a shared resource do not have their execution indefinitely postponed by mutual exclusion. A non-blocking algorithm is lock-free if there is guaranteed system-wide progress. The traditional approach to multi-threaded programming is to use locks to synchronize access to shared ressources. With a few exceptions, non blocking algorithms use atomic read-modify-write primitives that the hardware must provide. In our paper we want to examine and explain the most significant hardware primitives. Furthermore we want to illustrate usage of these by applying the primitives to a basic stack-algorithm to point out one of the most well known problems in this area - the ABA problem - which affects this kind of primitives. Finally we want to give an overview about traditional solutions and discuss them.

1 Introduction

First of all we want to give an overview and explanation about different properties in the area of lock-free algorithms.
Over the past two decades the research community has developed a body of knowledge concerning "Lock-Free" and "Wait-Free" algorithms and data structures. These techniques allow concurrent update of shared data structures without resorting to critical sections protected by operating system managed locks. Meanwhile there were many of different wait-free and lock free algorithms published, for simple data structures such as LIFO stacks and FIFO queues.

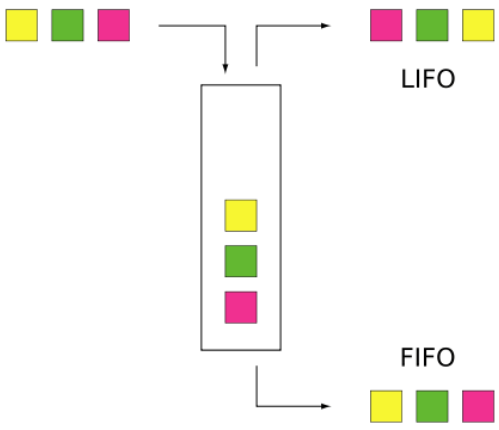


Fig. 1. Illustration of FIFO and LIFO stack/queue

FIFO and LIFO are acronyms for "First In, First Out" respectively "Last In, First Out". They are used in different areas. For example the FIFO acronym is well know in the usage of storage of food, where products are arranged in an order accordingly to their expire date. In computer science this terms refer to the way data stored in a queue is processed. The first item (data) which gets added to the queue will be the first item to be removed. We can associate the other acronym, the LIFO structure, with the well known term of a stack, where the topmost item, which is added last, is taken out first. The Figure above illustrates the two different structures where elements which get stored are taken out accordingly to the FIFO and LIFO semantics.

Lock-free algorithms for more complex data structures such as priority queues, hash tables, sets, and red-black trees are already known. So what are the most common benefits of lock-free synchronisation ?
First of all there are efficiency benefits compared to lock-based algorithms for some workloads, including potential scalability benefits on multiprocessor machines.
Further there exists support for concurrent update to shared data structures even when locks aren't available (e.g. in interrupt handlers etc.) and another advantage is the possibility to avoid priority inversion in real-time systems.

Before we are going to continue we want to explain the phrases Wait-freedom, Lock-freedom and Obstruction freedom. Wait-freedom is the strongest non-blocking guarantee of progress, combining guaranteed system-wide throughput with starvation-freedom.
An algorithm is wait-free if every operation has a bound on the number of steps it will take before completing.
It was shown in the 80s that all algorithms can be implemented wait-free, and many transformations from serial code, called universal instructions, have been demonstrated.

On the other side there is the phrase Lock-freedom. It allows individual threads to starve but guarantees system-wide throughput. An algorithm is lock-free if every step taken achieves global progress (for some sensible definition of progress). All wait-free algorithms are lock-free.
In general, a lock-free algorithm can run in four phases: completing one's own operation, assisting an obstructing operation, aborting an obstructing operation, and waiting. Completing one's own operation is complicated by the possibility of concurrent assistance and abortion, but is invariably the fastest path to completion.
The decision about when to assist, abort or wait when an obstruction is met is the responsibility of a contention manager. This may be very simple (assist higher priority operations, abort lower priority ones), or may be more optimized to achieve better throughput, or lower the latency of prioritized operations. Correct concurrent assistance is typically the most complex part of a lock-free algorithm, and often very costly to execute.

At last we want to explain the termin Obstruction-freedom. This is possibly the weakest natural non-blocking progress

guarantee. An algorithm is obstruction-free if at any point, a single thread executed in isolation (i.e. with all obstructing threads suspended) for a bounded number of steps will complete its operation. All lock-free algrithms are obstruction-free.
Obstruction-freedom demands only that any partially-completed operation can be aborted and the changes made rolled back. Dropping concurrent assistance can often result in much simpler algorithms that are easier to validate. Preventing the system from continually live-locking is the task of a contention manager. Obstruction-freedom is sometimes also called optimistic concurrency control.

Althoug Lock-freedom is not the strongest property and gives less guarantees than Wait-freedom, yet we are more interested in Lock-freedom. For several reasons Lock-freedom has some advantages in practice since it is not as costly to implement as Wait-freedom and its properties are sufficient to achieve special qualities for the functionality of distributed algorithms and shared data acess.
Significant benefits of Lock-free synchronization are that it avoids many serious problems caused by locks. These are things like considerable overhead, concurrency bottlenecks, deadlocks and priority inversion in real-time systems. Solutions for avoiding priority inversion usually involve special real-time process schedulers. On platforms where a real-time scheduler is not present, lock-free data structures provide an opportunity to sidestep the hazards of interlocking with the scheduler.
Almost all the lock-free algorithms require the use of special atomic processor instructions such as CAS (compare and swap) or LL/SC (load linked/store conditional). In our paper we want to examine and compare these two constructions. Furthermore we want to give a overview about the differences between primitives for lockbased and lock-free algorithms. Subsequently we want to explain shortly the main principles of this primitives.
Finally we want to denote the ABA-problem and its relation to the CAS (compare and swap) – instruction and how it cant be avoided by using LL/SC and its strong semantics.
Further we want to analyse the whole problem, point out the different solutions in a detailed way and discuss and evaluate them.

2 Atomic Processor Instructions

2.1 Compare and Swap (CAS)

The **compare-and-swap** CPU instruction ("**CAS**") (or the Compare & Exchange - CMPXCHG instruction in the x86 and Itanium architectures) is a special instruction that atomically compares the contents of a memory location to a given value and, if they are the same, modifies the contents of that memory location to a given new value. The result of the operation must indicate whether it performed the substitution; this can be done either with a simple boolean response (this variant is often called compare-and-set), or by returning the value read from the memory location. However the most common way is to return true or false. Figure 2 showes the state diagram of the CAS.

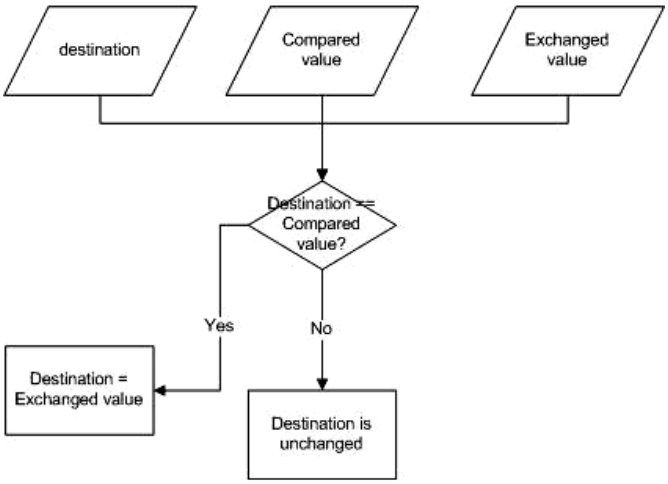


Fig. 2. Illustration of a CAS instruction

Here, described in Figure 3, want to give a description for the pseudocode of the CAS instruction. CAS compares here the content of the memory address with the expected value. If the comparison suceeds it replaces the content with the new value. The entire code of this procedure here is executed atomically.

```
template <class T>
bool CAS(T* addr, T exp, T val) {
        if (*addr == exp) {
                *addr = val;
                return true;
        }
        return false;
}
```

Fig. 3. Pseudocode of a CAS instruction

CAS is used to implement synchronization primitives like semaphores and mutexes, as well as more sophisticated lock-free and wait-free algorithms. Herlihy[5] proved that CAS can implement more of these algorithms than atomic read, write, and fetch-and-add. He stated if a fairly large amount of memory is used, CAS can implement all of them.
Algorithms built around CAS typically read some key memory location and remember the old value. Based on that old value, they compute some new value. Then they try to swap in the new value using CAS, where the comparison checks for the location still being equal to the old value. If CAS indicates that the attempt has failed, it has to be repeated from the beginning: the location is re-read, a new value is computed and the CAS is tried again.

CAS, and other atomic instructions, are sometime thought to be unnecessary in uniprocessor systems, because the atomicity of any sequence of instructions can be achieved by disabling interrupts while executing it. However, disabling interrupts has numerous downsides. For example, code that is allowed to do so must be trusted not to be malicious and monopolize the CPU, as well as to be correct and not accidentally hang the machine in an infinite loop. Further,

disabling interrupts is often deemed too expensive to be practical.
In multiprocessor systems, it is usually impossible to disable interrupts on all processors at the same time. Even if it were possible, two or more processors could be attempting to access the same semaphore's memory at the same time, and thus atomicity would not be achieved. The compare-and-swap instruction allows any processor to atomically test and modify a memory location, preventing such multiple-processor collisions.

2.1.1 CAS for nonblocking incrementation / Semaphores with blocking locks

Figure 5, which is an adaption of a figure given in [6], illustrates a comparison between a standard blocking algorithm and a non-blocking algorithm based on the use of a CAS instruction. The second variant of the algorithm provides a correct lock free code segment where different threads or processors just can try to increment the value of A.
The while-loop is running until the CAS instruction succeeds We recall that the CAS instruction is an atomic operation. Accordingly to the code, if several processors try to perform the INC function at the same time, there is always on processor which proceeds before all the others. It is the one which arrives first at the CAS codeline. The atomic operation applies a change to the shared memory adress A and because of this the CAS operation of all other processors will fail until they pass the loop again. This gets repeated until every processor performed its CAS operation successful which leads to the abortion of the while-loop.
The first variant in Figure 5 (blocking algorithm) can be achieved by using a semaphore which is a protected variable which constitutes the classic method for restricting access to shared resources (shared memory). Semaphores can be accessed using the following operations, shown in the following figure (Figure 4).

Lock:
P(Semaphore s) //atomic
{
< await until s > 0, then s := s-1 >
}
Unlock:
V(Semaphore s) //atomic
{
< s := s+1 >
}
Init(Semaphore s, Integer v)
{
s := v;
}

Fig. 4. Classical basic concept for semaphores

The incrementation of the variable s must not be interrupted and the P operation must not be interrupted after s is found to be greater than 0. This can be achieved by using instructions like **test-and-set** which must be supported by the instruction set of the given architecture.
It is quit obvious that this approach works fine but in practice semaphores are used in huge and complex code-structures and do not protect the programmer from producing faulty code which may cause the well known problem of a deadlock.

Standard blocking algorithm

```
proc inc(A)
lock
  tmp = A
  tmp = tmp+1
  A = tmp
unlock
end
```

Non-blocking algorithm

```
proc inc(A)
do
  tmp = A
until CAS(A, tmp, tmp+1)
end
```

Fig. 5. Illustration of blocking and nonblocking algorithm

2.1.2 CAS for nonblocking mechanism – Linked List

Another example for using the CAS instruction can be the following approach inspected in Overview of lock-free concurrency[3]. The CAS could be used to implement an algorithm for inserting elements into a shared linked list. For now we want to illustrate this just by showing a picture and giving a description of the activities.
Consider several processes, like illustrated in Figure 6, where each of them wants to insert an element into a linked list. Furthermore we assume an already existing structure with two elements which are contained in the list.

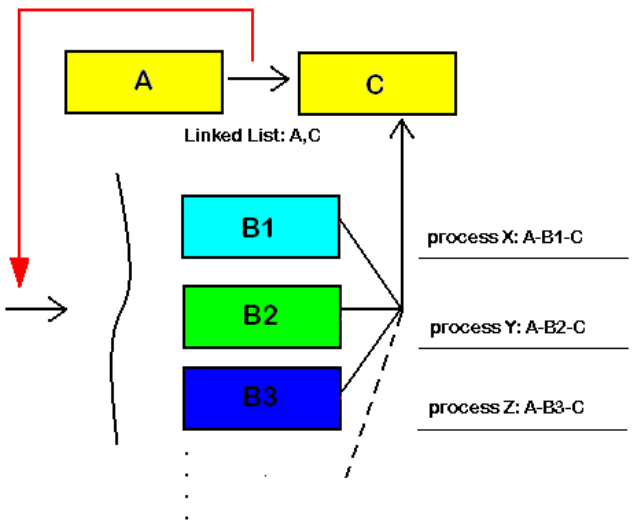


Fig. 6. Illustration of a linked list for concurrent access

The current state of the list holds two elements and can be written in the form A → C. Now consider the following actions for the processes. The first step for a process, lets say process X, is to set B1→ C. This step is still independent since the shared data structure (linked list) does not get changed.
No assume that another process, process Y, wants to insert also an element, lets call it B2. We further assume that several processors want to insert their element concurrently but we just consider this now in detail for process X and Y and their

element B1 respectively B2. The CAS instruction is here used to change the stored information that element A points on element C.
By using CAS the linking A → C should be replaced by some new reference like A → B1, A → B2 or A → B3 etc. Since all processes want to insert their element concurrently, one of them must be first. This depends on the fact which processer performs the CAS operation first. Finally we assume that process X will go first whereby the CAS of process Y will recognize that something has changed since the change happened atomically and for this reason Y can not proceed its CAS operation. Instead of that it will fail and try again in the next round. The same holds for all other possibly existing processes.
Even though the concept of just inserting elements into a list does not provide enough to be applicable it practice, it shows the principle of using CAS and finally we can conclude that it is possible to insert into a linked list concurrently with no locks.

2.2 Load-Link/Store Conditional

The instructions load-link (LL, also known as "load-linked" or "load and reserve") and store-conditional (SC) together implement a lock-free atomic read-modify-write operation.
Load-link returns the current value of a memory location. A subsequent store-conditional to the same memory location will store a new value only if no updates have occurred to that location since the load-link. If any updates have occurred, the store-conditional is guaranteed to fail, even if the value read by the load-link has since been restored.
As such, an LL/SC pair is stronger than a read followed by a compare-and-swap (CAS), which will not detect updates if the old value has been restored. This is an important property and we will discuss it in relation with the ABA problem in the next section again.

In the tight interpretation of LL/SC the instruction only fails if the referenced memory adress does not get accessed between the LL and SC command. But in quit a lot architectures also other events cause the LL/SC operation to fail and implementations of LL/SC do not always succeed if there are just no concurrent updates to the memory location in question. Exceptional events between the two operations, like a context switch, an interrupt or another load-link, or even (on many platforms) another load or store operation, will cause the store-conditional to spuriously fail..
This is often called *weak* LL/SC by researchers, as it breaks many theoretical LL/SC algorithms but weakness is relative, and some weak implementations can be used for some algorithms.

3 The ABA Problem

The ABA problem is a fundamental problem related to applications using CAS in lock-free and wait-free algorithms which are affected by this well known problem. In multithreaded computing, the ABA problem occurs during synchronization, when a location is read twice and has the same value for both reads, where the identical value is misleadingly used to indicate that nothing has changed. However, there is another thread which may execute between the two reads and change the value several times, do something and then change the value back to the original value. Later the original thread checks the value and erroneously proceeds under the asumption that the location has not changed even though the other thread did already perform some operation on it, which violates the assumption of the original thead. So the ABA problem occurs when multiple threads (or processes) accessing shared memory interleave. Below we illustrate a sequence of events that will result in the ABA problem:

1. *Process P_1 reads value A from shared memory,*
2. *P_1 is preempted, allowing process P_2,*
3. *P_2 modifies the shared memory value A to value B and back to A before preemption,*
4. *P_1 begins execution again, sees that the shared memory value has not changed and proceeds.*

Although P_1 can continue executing, it is possible that the behavior will not be correct due to the "hidden" modification in shared memory.
This problem is relatively strong related with the compare and swap (CAS) instruction mentioned before and the explanation for this is the following. It is possible that between the time where an old value is read and the time CAS is attempted, some other processors or threads change the memory location two or more times such that it acquires a bit pattern which matches the old value. The problem arises if this new bit pattern, which looks exactly like the old value, has a different meaning: for instance, it could be a recycled address.
If we go back now to our LL/SC (Load-Linked, Store-Conditional) instruction described in the section before, we can recall the ideal semantics of the atomic primitives LL/SC. They are inherently immune to the ABA problem. However, for practical architectural reasons, no processor architecture supports the ideal semantics of LL/SC. Current mainstream architectures support either CAS (Compare-and-Swap) or LL/SC with restricted semantics, which are not susceptible to the ABA problem but bring other problems. Furthermore, most current mainstream 64-bit architectures do not support atomic instructions on more than 64-bit memory blocks, thus making LL/SC implementations that require support for the atomic manipulation of wider memory blocks impractical.

3.1 ABA Problem illustrated by a stack algorithm

Figure 7 shows a variant of the IBM lock-free LIFO list (stack) algorithm [1] based on CAS which does not includes any mechanisms for preventing the ABA problem.
So let us consider a list which contains three nodes *A*, *B* and *C*. Now assume that a thread X reads the value *A* from the shared variable *Top* in line 4 and then proceeds to line 6 and reads the value *B* from A.Next.
After this it gets delayed. Then, another thread Y pops the node *A* from the list, then pops the node *B*, and finally pushes *A* again. Thus, leaving the variable *Top* with the value *A* and the list containing two nodes *A* and C. When X continues with its execution it proceeds until line 7 and the CAS instruction succeeds. The consequences of this CAS instruction are now setting *Top* to *B*. Finally this leads consequently to a corrupting list since B is actually not part of the list anymore and possibly this could also effect other structures, that may contain B, in a corrupt way. The intention of the algorithm designer is for X's CAS in line 7 to fail in such a case.
If it can be guaranteed that the CAS in line 7 cannot succeed if the value of *Top* has changed since the current thread's

execution of line 4, then the ABA problem would become impossible.
Besides there is to say that if memory does not get reused or if there exists a reliable garbage collection, the problem would not occur but since both properties are not always satisfied in lots of programming languages it remains a famous problem.

```
// Shared variables
Top:*NodeType; // Initially: null
Push(node:*NodeType) {
    do {
1:      t ← Top;
2:      node.Next ← t;
3:    } until CAS(&Top,t,node);
}

Pop() : *NodeType {
   do {
4:      t ← Top;
5:      If t=null return null;
6:      next ← t.Next;
7:    } until CAS(&Top,t,next);
8:    return t;
}
```

Fig. 7. ABA problem affected lock free LIFO list algorithm

4 SOLUTION FOR THE ABA PROBLEM

The earliest and simplest solution to this problem is the IBM tag methodology [1] shown in Figure 8. They use a tag (update counter) which is associated with the pointer *Top* (i.e.,the target of the CAS in line 7), such that when the pointer *Top* is changed the tag is also incremented atomically. By using double-width CAS, the pointer and the tag can be checked and updated simultaneously in one atomic step.
Especially we want to point out the change of datatype which is indeed necessary to implement this solution. Since we are using here double-with CAS, we need to redefine the typ definition for the pointer *Top* which is shown by the first line. Thus the following extension by an integer gets applied:

Top: *(NodeType,integer);

In terms of the semantics of the language C this would mean something like the following:

```
struct TOP {
        *NodeType node;
         integer value;
}
```

Treiber [2] pointed out that the change of the variables and the CAS instruction need only be applied to the Pop routine. For this reason Figure 8 just shows the modified pop routine which contains the IBM ABA-prevention tag mechanism.

Now we want to apply to the modified algorithm the same scenario, like the one in section 3.1 introduced before, which was applied to the CAS-based algorithm in Figure 8.

We consider again a list which contains three nodes *A*, *B* and *C* and we assume again that a thread X reads the value *A* from the shared variable *Top* in line 4 and then proceeds to line 6 and reads the value *B* from A.Next. Then X gets delayed and another thread Y pops the node *A*, then pops the node *B*, and finally pushes *A* again. Thus, leaving the variable *Top* with the value *A* and the list containing two nodes *A* and C.
In this special case the value of ***tag*** of *Top* has changed to ***tag + 2*** because of the two pop operations. Finally X continues with its execution and proceeds until line 7, but then the CAS instruction will be already able to identify that the counter associated with the pointer *Top* has beend modified. Accordingly to the algorithm in line 7 this means that the value of ***tag*** in <***t,tag***> does not match the integer value contained in the pointer *Top* anymore, since there have been several pop operations performed. In contrast to the previous algorithm, here the CAS instruction will fail.
The ABA problem can not occur and the list remains error-free. To be theoretical complete we assume that we have infinite integers for avoiding a wraparound of the tag.

```
Top: *(NodeType,integer); // Initially:  null, 0
Pop() : *NodeType {
   do {
4:      <t,tag> ← Top;
5:      if t=null return null;
6:      next ←  t.Next;
7:    }  until CASdbl(&Top, <t,tag>,<next,tag+1>);
8:    return t;
}
```

Fig. 8. LIFO stack algorithm using ABA-prevention tag

4.1 Solution with LL/SC based algorithm to avoid ABA problem

Here we want to present the solution based on LL/SC (Load-Linked,Store-Conditional) which bypass the ABA problem. If we recall our definition of LL/SC we have to consider that the strong semantics of the LL/SC instruction are used for implementing the following algorithm which is given by Figure 9.
In the algorithm LL takes one argument which is the address of a memory location and returns its contents. The SC instruction takes two arguments. The first one is the address of a memory location and the second one is the new value. Only if no other thread has written the memory location since the current thread last read it using LL, the new value is written to the memory location atomically. A boolean return value indicates whether the write occurred or not. The value returns *false* if any other thread has written the memory location since the current thread last read it using LL.
It is obvious and easy to recognize that if the basic write operation and the CAS instruction of the CAS based algorithm (Figure 6, line 4 and line 7) get replaced by the LL and SC instructions, the ABA problem can not occur anymore.
Thus the SC instruction in line 7 must fail if *Top* has been modified by some other thread since the current thread executed line 4.

```
// Shared variables
Top:*NodeType; // Initially: null
Push(node:*NodeType) {
    do {
1:      t ← LL(&Top);
2:      node.Next ← t;
3:    } until SC(&Top,node);
}

Pop() : *NodeType {
   do {
4:      t ← LL(&Top);
5:      If t=null return null;
6:      next ← t.Next;
7:    } until SC(&Top,next);
8:    return t;
}
```

Fig. 9. Lock free algorithm using LL/SC

4.2 Discussion of given solutions

Concerning the first solution, using a tag (update counter) associated to the pointer *Top,* the tag is assumed to have enough bits to make full wraparound practically impossible between a thread's execution of lines 4 and 7. So it depends on the number of bits which certainly can not be infinite. Although it is assumed for the moment that this issue does not bother anyone, anyway it will remain a theoretical problem which may cause a clash some day.
Another problem concerning this solution is that for making this work lock-free a DOUBLE-CAS (DCAS) operation is needed. This operation is currently only supported on x86 architectures by Intel and AMD CPUs for 32-bit mode but not supported in general which makes ist impossible to use it broadly. [4]
The second solution, using LL/SC, suffers from a general problem. As we mentioned before, this solution is restricted to the strong semantics of the LL/SC but in many architectures also other events cause the LL/SC operation to fail. For example interrupts or another load or store operation will cause the store-conditional to fail.
There are certain algorithms which can come along with this restrictions but in general it remains as a basic problem since the idea introduced in section 3.3 is in practice not applicable on many platforms.

5 Conclusion

In practice both, lock-free and lockbased algorithms, are used for solving different problems for accessing shared resources.
At first we pointed out the different approaches for implemeting lock-free and lockbased algorithms and we also illustrated their primitives. We gave an explanation for using these primitives and pointed out the main difference between them and further the drawbacks of primitives for lockbased algorithms.
The well known problem of a deadlock is one of the major disadvantages in the field of lockbased mechanisms and it was shown easily, how this problem can be avoided by using lock-free processor instructions. Moreover we introduced this kind of instructions, which are the CAS and LL/SC instructions and gave a detailed explanation about their functionality and pointed out their differences. Mainly we focused on CAS, where we provided two different illustrations, using the CAS instruction for accessing shared resources in a lock-free way.
Furthermore we introduced the ABA problem and how it is related to the CAS instruction. Therefore we showed the basic stack algorithm[1] and explained its susceptibility for the ABA problem. Subsequently we pointed out two well known solutions illustrated by one modifed and another new algorithm[1]. On the basis of that we showed that it is possible to avoid the ABA problem either with using DOUBLE-CAS or LL/SC.
Finally we can conclude that both solutions suffer from different problems. The minor problem of the CAS based solution is related to the fact of unbounded interger values and as far as we can think, it remains a theoretical problem.
But in general both solutions are affected by the fact of insufficient support of processor architectures, since the ideal semantics of LL/SC and the DOUBLE-CAS are not very common. Thus, for broader applying of lock-free algorithms in general, it may help to develop more of these specific needs, when building new processor architectures.
We want to recall our statement that in most of the applications the Lock-freedom property, which is weaker than Wait-freedom, is sufficient for establishing accurate access to shared resources. Besides we want to mention that CAS and LL/SC cannot provide starvation-free implementations of many common data structures without memory costs growing linearly in the number of threads. Wait-free algorithms are therefore rare, both in research and in practice.

References

[1] IBM. IBM System/370 Extended Architecture, Principles of Operation,1983. Publication No. SA22-7085.

[2] R. K. Treiber. Systems programming: Coping with parallelism. Technical Report RJ 5118,IBM Almaden Research Center, Apr. 1986.

[3] Overview of lock-free concurrency, http://burtleburtle.net /bob/hash/lockfree.html

[4] Jean Gressman, The ABA Problem explained, http://fara.cs.uni-potsdam.de/~jsg/nucleus3.23/ index.php?itemid=6

[5] M. P. Herlihy. A methodology for implementing highly concurrent objects. *ACM Transactions on Programming Languages and Systems*, 15(5):745–770, Nov.1993.

[6] Introduction to non blocking algorithms, Alexandre David, www.cs.aau.dk/~adavid/teaching/MVP-08/06-Non-blocking% 20algorithms.pdf

Lock-Free Hash Table Implementations

Jasper Smit

Abstract—
We compare two algorithms which implement a lock-free hash table to be used for concurrent systems. Shalev and Shavit[8] introduced a method based on direct chaining. Gao e.a.[3] also proposed an implementation for lock-free hash tables. His approach is based on open addressing. We compare both methods and briefly explain the algorithms.
For the comparison we look at implementation details and to the complexity properties of the algorithm.
It turns out that the algorithm of Shalel and Shavit is prefered for it's better elegance, simplicity and better worse-case performance.

Index Terms—Hash tables, Distributed algorithms, Lock-free

1 Introduction

A hash table is a data structure which is used to store items which are identified by an unique key very efficiently. Traditional hash table implementations can usually not be used by multiple threads. The thread safe variants of hash tables are typically realized by mutual exclusion. In this paper we compare two different techniques to implement a lock-free hash table. The first techique is described by Shalev and Shavit [8]. A totally different approach is taken by Gao e.a.[3]. In this paper we discuss both techniques and compare the algorithms. We look at the differences in the two algorithms and look at complexity and other details.

In the first part we will explain traditional hash tables. Next the hardware primitives for concurrency are explained. Then we will discuss both approaches. In the final part of the paper the two approaches are compared with each other.

2 Hash Tables

A hash table consists of a large array[2]. The elements of this array are called buckets. Depending on the implemenation one or more data items can be stored in a single bucket. Usually many of the buckets are empty. The proportion of the hash table which is filled with items is called the *Load factor*. A hash functions maps the keys of items to be stored to one of the buckets. Evaluation of this function takes $O(1)$ time. Thanks to this the bucket where an item belongs can be found in constant time and therefore finding and inserting can be implemented very efficient. These operations only take constant time.

In most situations it is not possible due memory limitations to use the same number of buckets as there are possible keys. Therefore a hash function has to be chosen which maps multiple different keys to the same bucket. This can lead to *collisions* when more than two items with different keys map to the same bucket. Different implementations use other ways to deal with collisions. The two most popular collision resolve techniques are *open adressing* and *direct chaining*. These techniques add extra overhead to the hash table operations, but when the number of collisions are limited they can still be performed in constant time.

2.1 Open addressing

When inserting a new item causes a collision in an open addressing hash table another free bucket, is selected for that item. One way to select a new candidate bucket is *linear probing*. At linear probing the algorithm keeps visiting adjacent buckets until a free bucket is encountered. If bucket *i* is occupied, it tries the next bucket or if the last bucket is occupied it tries the first. The item will be stored in the first free bucket.

Since items can possibly be stored at different buckets than the hash function maps it to, it also requires a changes in the find and delete operations. The find operation of an open addressing hash table does a linear search in the hash table, starting at the bucket where the found key was mapped to. It continues looking until a free bucket is reached.

When deleting an item, a bucket cannot simply be emptied. This free bucket can prevent the find operation to find an item. The find operation keeps searching until it encounters a free bucket. A solution during deletion is to move all the items with the same hash, comming next to the deleted item one bucket down. Then it fills the empty spot of the deleted item and all items which share the same hash are adjacent and can still be found. It is also possible to mark the bucket as 'deleted'. This ensures that the find operation does not stops it linear search at that point. However a long sequence of insertion and deletion operation will bring the hash table to a state where all of the buckets are marked as deleted. Find and insert operations will than take $O(n)$. It is therefore necessary to *rehash* the table from time to time. When the number of marked buckets gets too high all stored items will be moved to another hash table and the old one is abandoned. The marked buckets are not moved.

A hash table with open addressing is illustrated in Figure 1.

0	Item A (hash 0)
1	
2	Item B (hash 2)
3	Item C (hash 2)
4	
5	
6	Item D (hash 6)
7	*Marked deleted*
8	Item F (hash 6)
9	

Fig. 1. A hash table with open addressing. When collisions occur, items with the same hash get stored in adjacent bucket. In this figure item E has been deleted. Bucket 7 cannot be emptied, because in that case item F with hash 6 cannot be found anymore.

2.2 Direct chaining

The direct chaining approach (see Figure 2) allows more items to be stored in the same bucket. This can be implemented by realizing each bucket with a secondary data structure like a linked list.

Direct chaining hash tables have as advantage that the operations, especially the delete operation are simpler and there is no need to rehash the table, because it does not get poluted due to many deletions like in open addressing. Also the performance degrades more smoothful when the load factor gets higher.

As a downside the direct chaining approach suffers from bad cache performances, due to bad locality of reference of the linked lists at each bucket.

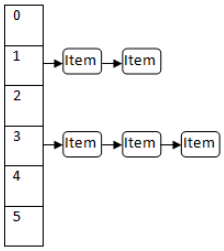


Fig. 2. A hash table which uses direct chaining. Items with the same hash are stored in the same bucket.

2.3 Resizing

A hash table has a fixed number of buckets. When the number items contained in the hash table gets too high its performance will degrade. In open adressing hash tables there is a hard limit, since the number of items cannot exceed the number of buckets.

For these reasons the hash table has to be resized when the load factor gets too high. The resize operation is done when an insert causes the load factor get above a certain threshold. This threshold is usually set around 70 or 80%.

When the hash table has to resize, a new hash table is allocated, a new hash function is chosen and all items currently contained in the old hash table are moved to the new hash table.

Resizing a hash table is a very expensive operations. It takes $O(n)$ time to allocate the new memory and to move all the items. Fortunatly this does not have to be done on every insertion and therefore the amortized time complexity for insertion is still $O(1)$, if the resize is done in an efficient way. This constant amortized time complexity cannot be achieved when the size of the hash table is increased by one at every resize operation, however when doubling the size at each resize this constant amortized time complexity can still be achieved.

2.4 Lock-free datastructures

Without extra care most datastructures cannot be used by multiple processes concurrently. If more processes mutate the datastructure, race conditions can occur. Due to race conditions it could happen that operations performed by one process are discarded by the interference of another. It is even possible that the data structure gets corrupted when two processes modify the hash table at the same time. Consider for example two concurrent insert operations in a hash table. Both inserted items happen to be in the same bucket. The two processes observe at the same time that the bucket is free, then they both insert their item at the same place, leading to the situation that one of the inserted items is overwritten and thus lost.

2.5 Existing lock-free datastructures

Special concurrent implementation of datastructures like the hash table are available. They are usually implemented by realizing mutual exclusion. If only one process can access a the datastructures at the time there is no way race conditions can occur. To realize the mutual exclusion some critical section have to be locked.

The use of locks however causes several problems. The lock is a performance bottleneck. When one process is accessing the hash table all others are forced to wait, making the system not truely concurrent anymore. Furthermore, a failure of a proces that has locked the hash table makes the complete system reach deadlock.

For this reason concurrent algorithms which do not use locks are investigated. Several lock-free algorithms have been proposed[1, 9, 10].

2.6 Concurrent primitives

Almost all of the lock-free algorithms depend on hardware support for certain atomic operations. The most used atomic hardware operations are discussed in this section.

2.6.1 Fetch-and-add

The fetch-and-add operation increases the value of a specified memory location by one and then it returns the increased value. Like the fetch-and-add there is also a fetch-and-decrease.

2.6.2 Compare-and-swap

Compare-and-swap or Compare & Exchange is an operation which writes to a specific memory location only if this address contains a specified value. The check wether the memory location contains the value is done atomically. It is guaranteed that no other processes modify the location between check and the write. The operations returns wether a write has taken place or that the value contained on the memory location was different than expected.

The compare and swap can be summarized by the following code executed atomically:

```
bool compare-and-swap(*address,
    expected_value,
    new_value) {
  success = *address == expected_value;
  if(success) *address = new_value;
  return success;
}
```

The compare-and-swap is implemented in most modern architectures. It is supported by the x86 processor since the 486. Most implementations of the compare-and-swap work on word-sized locations only, so bigger sized memory locations cannot be replaced atomically. There are extensions on some architectures which can work on double words, called the double-wide-compare-and-swap. This operation is implemented on modern x86 architectures. Another extension is the double-compare-and-swap which works on two memory locations and only writes two suplied new values when they are both equal to the two specified expected values. Some efficient lock-free algorithms are based on the double-compare-and-swap operation. At the moment there are only a few CPUs that implement this operation.

Maurice Herlihy[5] proved in 1991 that using compare-and-swap operations more concurrent algortihms can be implemented than ordinary atomic read/writes and fetch-and-add.

2.6.3 Load-Link/Store-conditional

Load-Link and Store-Conditional are two operations which are used together. These are even stronger than a compare-and-swap operations. A load-link operation loads a value from memory. After this value is loaded and some operations are done with it the store conditional is used to store a changed value at the same location. The store conditional fails if from the time that the value was read with load-link has changed, even if it has changed first and then original read value was restored. Without changing the writing process this behaviour is not realizable by only using compare-and-swap.

3 The algorithm of Shalev and Shavit's

In their article 'Split-Ordered Lists: Lock-Free Extensible Hash Tables'[8], Ori Shalev and Nir Shavit propose a lock-free hash table based on direct chaining. The hash table supports resizing when the internal load factor gets too high. The resize operation is done without using locks on the table or on individual buckets.

The hash table consists of a single linked list, as illustrated in Figure 3. Each bucket of the hash table is a part of the linked list. The buckets link to a node of the linked list. These nodes are called dummy nodes, because they do not store actually data items. Between the dummy nodes are data nodes which do have an item attached to them. There can be more than one item stored between two dummy nodes, realizing the direct chaining of the hash table.

Items inside the list should be ordered by their keys. This makes it possible to later add more dummy nodes in between two existing dummy nodes.

Insertion and deletion are implemented with corresponding linked list implementations of insert and delete. Shalev and Shavit use a lock-free linked list implementation from the work of Michael [7]. This implementation uses compare-and-swap primitives. The delete and insert operations first revert the key before delegating the task to the linked list operations. The hash function used for this implementation is: `hash = key % size`. This is necassary for the resize operation to work, the details are explained in the next section.

3.1 Resizing

When the number of items inside the hash table gets higher than the maximum load factor the number of buckets in the hash table will be doubled. In traditional hash table implementations all items will be redistibruted among the buckets.

Shalev and Shavit do this the other way around, the buckets get redistributed among the items. After each existing dummy node a new dummy node is inserted. In the array the new buckets are just inserted after the existing buckets. The hash function outputs an aditional significant bit. This extra bit is the most significant bit of the array index. The index of the linked list is the same key, but then reversed binary. This way an extra least significant bit is added to the linked list key, effectively splitting each segment in half. The new dummy nodes can then be inserted in the existing list without having to rearrange all the items.

The situation after resizing is illustrated in Figure 4.

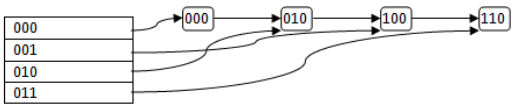


Fig. 3. A hash table with 4 buckets, only displaying dummy nodes. Note that the key of an item is reversed in the list index, explaining the cross-over in nodes 01 and 10.

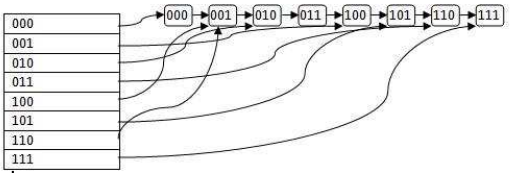


Fig. 4. The same hash table as in Figure 3, but grown to 8 buckets. In the bucket list a higher order bit is added to the key, while in the linked list the reverse key is used adding a bit to the right, creating the effect of new buckets getting in between the old ones.

Unfortunatly there is no way for the hash table to shrink again. When the hash table has reached some worse case capacity, from that moment it will always be at that size.

3.2 Finding items

The find operation first calculates the hash. This hash is the bucket which points to a dummy node in the linked list. It then traverses the linked list beginning at that dummy node. Because the linked list is ordered on key it can keep on searching until a node is encountered which has a higher key.

3.3 Insertion

The insert operation does an insert in the linked list. If the item was not yet contained in the hash table, the insertion is considered succesful. In that case the number of items counter is incremented, using fetch-and-add. With this counter the load factor can be calculated. When the load factor gets too high, the hash table grows by doubling the size of the bucket array. This can simply be done by doubling the size variable, because the buckets are initialized lazy. After this variable has doubled, the hash function (hash = key % size) distributes the items over twice as many buckets. The first time a bucket is refered to it will get initialized and inserted in the linked list at the right location.

3.4 Deletion

The delete operation finds the correct bucket and from this point it calls the delete operation of the linked list. After the item is deleted the item counter is decremented atomically.

4 The algorithm of Gao e.a.

Gao, Groote and Hesselink proposed in their article 'Lock-free Dynamic Hash Tables with Open Addressing' [3] another method for lock-free hash tables. Their approach differs from Shalev and Shavit's approach by using open addressing instead of direct chaining. When resizing the table all processes currently accessing the table helps migrating items from the old table to the new one. The new table does not necassarly have to be bigger than the existing table. This depends on the number of items marked as the deleted, if the current table contains many of these items, the table actually becomes smaller.

4.1 Data Structure

Because each process can decide to allocate a new hash table for migration, multiple hash tables can exist at the same time. A shared array `H` is used to store all hash tables. A shared variable `curInd` indicates which hash table is the active one.

An array `next` is used during migration. If `H[i]` is a hash table which is currently migrating to a new hash table, then the new hash table is indicated by `H[next[i]]`. Another array called `busy[i]` is a counter of how many processes are accessing hash table `H[i]`. `busy` is not part of the hash table, but stored in a separate array. Because when `busy[i]` is zero it could happen that the hash table is about to be thrown away. When at that moment a process increments the `busy` variable it will write on deallocated memory. Another array `prot[i]` is used to protect the reuse of the variables `H[i]`, `busy[i]`, `next[i]` for a new hash table when other processes still use these values.

4.2 Migration process

An insertion can initiate a migration when the insertion causes the number of buckets too get higher than a certain threshold, called the boundary. The process then allocates a new hash table. The size of the new hash table is determined by the current boundary and the number of items marked for deletion. This latter number decreases the size of the created hash table, allowing it to shrink when many items are deleted. The new size has to be at least larger than `bound + 2 P`, with `P` being the number of processes. This is necessary, because when resizing there can still be `P - 1` pending insertions from other processes.

After the process has allocated for the original hash table `H[i]` a new resized hash table `H[j]`, it assigns using a compare and swap `next[i] = j`. If `next[i]` was already assigned a value, the compare-and-swap fails. The procces can then deallocate the hash table for another process was first in creating the new hash table.

When the new hash table has been prepared, items can be migrated from the old hash table. The process allocating the new hash table begins with this job. A single item to be migrated is first marked as old, so that other processes in the migration process do not start copying the same item. Old items can still be found by concurrent find operations. Next the item is inserted in the new hash table and finally the item in the old one is deleted, by replacing it with a special `done` item. If concurrent delete or insert operations detect these old or done elements, the process knows the current hash table is being abondend. It then joins the migration process, starting moving elements to the new hash table. Find operations can still use the values marked with old, but if they are replaced with `done` they also start joining the migration process. After all items are migrated, all buckets in the old hash table have been replaced with `done` and it is therefore guaranteed that they will not be used by other processes anymore.

4.3 Hash table reference

Before a process can perform any operation on the hash table it first has to get a reference to the hash table, by calling `getReference`. This function copies the shared variable `curInd` to a private variable `index`. It also increases `prot` and the reference counter `busy`. It can then do one or more operations on the hash table until it calls `releaseAccess`, which decreases the reference counters. `releaseAccess` is also responsible for releasing the memory of abandoned hash tables. When `busy` has dropped till zero, the last process releases the memory of the hash table.

Although `getReference` sets the `index` for the current process, this value can change during the execution of operations on the hash table. This happens when a migration has started.

5 Comparison

We have discussed two methods to implement a lock-free hash table. Although they both do the same thing, the implementations are quite different. In this section the methods of Shalev and Shavit are compared with the algorithm of Gao e.a.. We will look into the differences and similarities in implementation and the up and down sides of the discussed algorithms.

The algorithm of Shalev and Shavit is much simpler and elegant than the other. The algorithm of Gao e.a. is a very complicated algorithm which is hard to follow. This makes implementation difficult and error prone. The algorithms for the operations are much longer. For each access to the hash table an indirection step is necessary to pick the correct hash table from an array.

Both Gao e.a. and Shalev and Shavit provide a proof for the correctness of their algorithm. For Shalev and Shavit it is intuitive to see that the algorithm is correct. In his paper he provides only a skeleton proof. For the algorithm of Gao e.a. it is not directly clear why the algorith is correct. Gao e.a. have proved their algorithm completely, using mechanical proof verification. The proof is very long, it consists of about 200 invariants. It took them about two men year to complete the prove. In the original paper the safety and progress proofs are explained.

5.1 Simalarities

Both algorithms rely on the implementation of atomic operations in hardware. These operations are absolutly necessary, without these primitives it would be impossible to implement the algortihms, as shown by Herlihy [5]. The most used atomic operation by the algortihms discussed is compare-and-swap. Fetch-and-increase and fetch-and-decrease are also both used by the algorithms to implement counters, like the number of items contained or the number of deleted items.

Another simalarity is that both algorithms support upsizing the hash table when necessary. Algorithms for non resizing lock-free hash tables were already described earlier like [7, 4].

5.2 Shrinking

The method of Shalev and Shavit does not support shrinking, where as the other method described by Gao e.a. can shrink when many items are deleted. This means for Shalev and Shavit's method that when the hash table has grown huge because of some peak of elements to be inserted, it will never shrink to normal proportion even if the number of items is marganalized. The hash table of Gao e.a. better adjusts to its current memory needs. This might not be a big matter, because it is shown that in practice hash tables only need to increase in size [6].

5.3 Resizing

Although hash tables of both types can grow, they both grow in different fashions. For Shalev and Shavit growing just means increasing a variable. This does not take considerable amount of time. Because the new buckets are kept unitialized until they are refered, the resize operations overall can be done very efficient in $O(1)$. Other processes do not have to wait before resize operations is completed, they can still use the old buckets.

The resize operations of Gao e.a. takes $O(n)$ which is significantly slower than the other approach. During a resize a complete new hash table is allocated and all items in the old hash table has to be moved to the new hash table. In worst case scenario it can even take $O(n^2)$, this happens when all participating processes pick the same hash table elements to migrate.

Algorithm:	Shalev & Shavit	Gao e.a.
Elegant algorithm	Yes	No
Supports resizing	Grow only	Grow and shrink
Resize time complexity	$O(1)$	$O(n)$, worst-case $O(n^2)$
Memory complexity	$O(m)$	$O(m)$, worst-case $O(Pm)$
Processes can join	Yes	No
Easy to understand	Yes	No

Table 1. Comparison of both algorithms

Concurrent operations during a resize have to join the migration process and have to complete this before their operation can be completed. If all processes run at the same speed there is no advantage for the process to join in and help over blocking and letting one process do all the work. Due to the memory bottleneck all processes have to take turn to write to the memory anyway.

5.4 Efficiency

Traditional hash table implement all operation, like find, insert and delete in $O(1)$. Resizing a hash table takes about $O(n)$. Despite that the amortized time complexity is still $O(1)$ for each operation, because a resize is only necassary in rare ocassions.

For the lock-free variants of the hash table this complexity still holds. The time complexity of resizing is already discussed in the previous section.

The memory complexity of both hash tables are under normal conditions $O(m)$, m being the number of buckets in the hash table. However the algorithm of Gao e.a. can be under worst case conditions be $O(Pm)$, with P being the number of processes. This happens when all processes decide to allocate a new hash table at the same time.

Looking at the time and space complexity of boths algorithms, the hash table implementation of Shalev and Shavit performs better under worst case conditions.

5.5 Joining processes

The matter of joining or leaving processes is not mentoined in either of the two papers. For Shalev and Shavit's implementation it is no problem that other processes join or leave, nowhere in the algorithm a constant number of processes is assumed.

In the implementation of Gao e.a. the number of processes is fixed. This limitation is due to the design of the data structures. These contain arrays of which the length is dependent on the number of processes. If another process joins, more positions should be made available in the array. Also at resizing the hash table the new size of the hash table is based on the number of processes. This problem however is easily fixed, by taking initially a larger array than the number of processes. Now the number of processes can grow during the executing until a certain maximum. Leaving processes is not a problem in either of the algorithms

6 Conclusion

In Table 1 the comparison is summarized. The algorithm based of Shalev and Shavit should be prefered. This the simplest and most elegant of the two algorithms. Both the implementations support growing, but only Gao e.a.'s hash table can shrink. Under normal conditions the algorithms have the same time and memory complexity as a traditional hash table implementation. Shalev and Shavit support joining processes later in the execution of the algorithm. The algorithm of Gao e.a. has a worsed case complexity which is not as good as Shalev and Shavit's.

References

[1] G. Barnes. A method for implementing lock-free shared-data structures. In *SPAA '93: Proceedings of the fifth annual ACM symposium on Paral-*

lel algorithms and architectures, pages 261–270, New York, NY, USA, 1993. ACM.

[2] J. Erickson. Lecture about hashtables. http://compgeom.cs.uiuc.edu/ jeffe/teaching/373/notes/06-hashing.pdf.

[3] H. Gao, J. F. Groote, and W. H. Hesselink. Efficient almost wait-free parallel accessible dynamic hashtables. *CoRR*, cs.DC/0303011, 2003.

[4] M. Greenwald. Non-blocking synchronization and system design, 1999.

[5] M. Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, 1991.

[6] M. Hsu and W. Yang. Concurrent operations in extendible hashing. In W. W. Chu, G. Gardarin, S. Ohsuga, and Y. Kambayashi, editors, *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*, pages 241–247. Morgan Kaufmann, 1986.

[7] M. M. Michael. High performance dynamic lock-free hash tables and list-based sets. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 73–82, New York, NY, USA, 2002. ACM.

[8] O. Shalev and N. Shavit. Split-ordered lists: lock-free extensible hash tables. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 102–111, New York, NY, USA, 2003. ACM.

[9] J. D. Valois. Implementing lock-free queues. In *In Proceedings of the Seventh International Conference on Parallel and Distributed Computing Systems, Las Vegas, NV*, pages 64–69, 1994.

[10] J. D. Valois. Lock-free linked lists using compare-and-swap. In *In Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 214–222, 1995.