# Experimental User Interface Design Toolkit for Interaction Research (IDTR)

Jean Claude Raymond Rolin Golovine

PhD 2013

# Experimental User Interface Design Toolkit for Interaction Research (IDTR)

Jean Claude Raymond Rolin Golovine

A thesis

submitted in partial fulfilment of the

requirement of the

Robert Gordon University

for the degree

of Doctor of Philosophy

**May 2013**

# Abstract

The research reported and discussed in this thesis represents a novel approach to User Interface evaluation and optimisation through cognitive modelling. This is achieved through the development and testing of a toolkit or platform titled Toolkit for Optimisation of Interface System Evolution (TOISE). The research is conducted in two main phases. In phase 1, the Adaptive Control of Thought Rational (ACT-R) cognitive architecture is used to design Simulated Users (SU) models. This allows models of user interaction to be tested on a specific User Interface (UI). In phase 2, an evolutionary algorithm is added and used to evolve and test an optimised solution to User Interface layout based on the original interface design. The thesis presents a technical background, followed by an overview of some applications in their respective fields. The core concepts behind TOISE are introduced through a discussion of the Adaptive Control of Thought – Rational (ACT-R) architecture with a focus on the ACT-R models that are used to simulate users. The notion of adding a Genetic Algorithm optimiser is introduced and discussed in terms of the feasibility of using simulated users as the basis for automated evaluation to optimise usability. The design and implementation of TOISE is presented and discussed followed by a series of experiments that evaluate the TOISE system. While the research had to address and solve a large number of technical problems the resulting system does demonstrate potential as a platform for automated evaluation and optimisation of user interface layouts. The limitations of the system and the approach are discussed and further work is presented. It is concluded that the research is novel and shows considerable promise in terms of feasibility and potential for optimising layout for enhanced usability.

*Dedicated to my father, my mother and lost daughter Kirsteena-Lee and all my friends and family who are sadly no longer with us today*

# Table of Contents

# Table of Figures

# Table of tables

# Table of Abbreviations

# Acknowledgement

This study would not have been possible without the support of my family at home. My partner Debra supported me throughout all those years, running the house with her determination to keep the family together and reminding me of the important things in life. This thesis would not have been possible without her help, love and her patience and for putting up with me all those years. I would like to thanks the children, Jack for his care about my work, Ellen for being understanding and Stephane for always being supportive when things became difficult.

I would like to thank the Robert Gordon University and my fellow colleagues for all the support and encouragement they provided to me, with a special thanks to Audrey Fryer for the all so helpful coffee sessions. Many thanks to the technical team that have provided their professional assistance on so many occasions well beyond expectations and to Malcolm Clark for going over this thesis.

Most importantly, this thesis would not have achievable without Patrik O'Brian Holt, who gave me this opportunity, provided me with the support, mentorship, knowledge and lively chats without which this research would never have been completed. Special thanks also go to John McCall for keeping me on track and providing me with ideas that helped to bring this project to completion.

# Chapter 1. Introduction

While the fundamental lessons of Human Computer Interaction (HCI) as a field of study is important, the significance of user experience design in the interface design and engineering should not be under estimated. Users are often faced with interfaces created by software engineers. Theses engineers are highly trained specialists, who tends however to focus less on aspects of user interaction than they are on the software functionality. It is often the case that users tend to be seen as secondary participants in the application development process, who are often considered once the software development life cycle is nearing completion, which is after the functionality is tested. It is most often seen that users need to conform to the requirements of the application and its interfaces created by the developers, as opposed to the idea of the design of the application being a collaborative activity between those users and the system developers.

As new technologies such as new mobile innovations impact on our daily life, a new type of computer user interfaces and users emerges. Today's computer interfaces have become much more interactive, complex and ubiquitous in many devices, and users spent much more time interacting with them as part of their day-to-day lives. For example, not only these users utilise this technology as tools as part of their professional lives but often in their private life too for leisure, such as communication and entertainment. This new generation of users and devices highlights the necessity to develop software and interfaces fit for purpose for the highly diverse population of users.

HCI has emerged alongside computer technologies as it soon became apparent that sophistication of software is pointless unless the users can interact properly with its interfaces. When software is designed, it is embedded with functionality that is only perceived when users use the system efficiently and represents what this system can do and above all what it is marketed for. The usability of this software is the degree by which it is used in terms of efficiency when users perform a series of possible tasks using the system. Thus as Karray stipulates, a balance is needed between the actual functionality and the usability to demonstrate efficiency (Bevan 1998; Karray et al. 2008).

Usability has potential benefits for software development companies such as increased productivity and customer satisfaction (Haapasalo and Kess 2002; Rajanen and Jokela

2007; Kljajevic 2008) as it brings effectiveness and efficiency to the product. It is increasingly significant for leading world software companies to use usability factor in high profile advertising (Karat 1993; Myers 1993; Byrne 2008; Macleod 1994; McGrenere 1998; Gamsriegler 2006). As Macleod states, usability tends to be linked with minimization of errors and staff training, efficient use of software and lower staff turnover, which stir Usability Evaluation (UE) as an essential part of the development to ensure that user interfaces (UIs) enable users to achieve goals effectively in line with Standards such as ISO9241 (Nielsen 1994; Bevan 1998; Lebiere et al. 2005; Lane 2007; Ismail et al. 2011). As Haapasalo and Kess stipulate, it is more likely that people operates computer systems to be productive but not essentially to be creative and thus, software development is often driven around routine tasks (Redmond & Smith 1975; Haapasalo & Kess 2002; Lif et al. 2003; Anderson et al. 2004). McGrenere noted that most users are now faced with more functionality in their computerised systems and marketing was often perceived as the driving force behind this increase of features even though surveys report that heavily featured applications are often the cause for a decrease of productivity. This statement is more accurate than ever today with consumerism having become a major part of our daily life and users purchase computer software and electronic devices in the same manner as they would with any other products i.e. they want more features and functionality for their money even though it is unlikely those devices extra functions that are often heavily marketed will ever be used by the majority of users (Sutherland 1964; Lewis & Rieman 1993; Anderson 1996; McGrenere 1998; Lebiere 2006).

HCI research has made the case for usability evaluation as part of the design process that leads to interface optimisation. However, given the known potential benefits, there are comparatively few software developments that undertake usability testing as part of their software development life cycle, as often management experience difficulties to establish immediate potential advantages. In addition, another aspect is that usability evaluation often competes in terms of budget share with other development teams (Karat 1993; Anderson & Lebiere 1998; Troutt & Sheiner 2007). Many software designers refrain from using usability as an iterative process in their software development as the process is perceived as too time consuming and expensive; especially regarding the techniques involved too intimidating in their complexity (Newell 1990; Dillon et al. 1994; Nielsen 1994; Lif et al. 2003; Das and Stuerzlinger 2007; Pang & Schauder 2007; Ismail et al. 2011).

Most large projects are developed using version of the waterfall method, which decomposed the entire process into a series of well-defined sequential phases. It starts

with the requirement analysis, and then a set of specification documents is drawn for it. The project is then planned and the design phase starts. Once completed, the implementation is undertaken; with a phase of integration, testing and maintenance that ends the project development. In this model there is no formal mechanism for feedback from one phase of the development to previous phases which often impairs maximising usability.

Even though this model has been proven to fail in many cases it is still used as it allows companies to define responsibilities and budgets easily. This state of affair has major consequences as testing is achieved by a very limited sample of potential users which can be problematic when an attempt of optimization of interfaces is necessary (Lewis & Rieman 1993; Kieras 1997).

Therefore, usability in software engineering is often neglected despite over thirty years of research discussion on the HCI scene (Nygaard & Dahl 1981; K Card et al. 1983; John et al. 2004; Troutt and Sheiner 2007). Cost is often always the limiting factor rather than the belief that usability is being considered un-beneficial, as it is known that well-designed software increases productivity, reduces human errors and makes software more enjoyable to users, which in terms of marketing is an important point. Dillon (1994) surveyed current practices in usability engineering and found that those benefits could be measured by a reduction of training and learning time most feedback from the professionals demonstrated a view that reflected a vague nature of the concept. However this survey also reported that 52% of the respondents reported that their organisation perceived usability as “Essential” while 19% of the respondents viewed it as unnecessary. 95% suggested that a solid knowledge of the users, tasks and their working environment was very important suggesting most of the respondents had a realistic opinion in terms of usability and related variables. However, 64% of them reported having difficulties with usability because of limited time and resources (Dillon et al. 1994; Dorman and Gaudiano 1998; Ivory and Hearst 1999; Ivory and Hearst 2001). This statement is still valid today.

There is however another school of thought that has emerged and rejects the traditional waterfall development path and has moved in a more participatory design model where systems are designed by both the designers and users working together with the common goal of making the system fit for purpose (Lebiere et al. 2005; Pang and Schauder 2007; Bolton et al. 2012), in effect moving the process from “designing for” to “designing with” (Card et al. 1980; Lewis and Rieman 1993; Anderson et al. 2004).

To improve the undertaking of usability evaluation many HCI specialists have responded by providing specialised tools and usability guides to designers to validate and evaluate their designs in an attempt to optimise them to aid designers to assess their software interfaces at low cost both in terms of modelling and financial, thus promoting usability (Anderson 1996; John et al. 2004; Grudin 2005; Lebiere 2006).

Automated tools are increasingly been researched (Card et al. 1983; Ivory & Hearst 1999; Ivory & Hearst 2001; Crystal and Ellington 2004), but automatic optimisation of UI is a field that remains undeveloped.

The goal of the research that is presented in this thesis is to provide tools for design and research titled: *Toolkit for Optimisation of Interface Systems through Evolution (TOISE).* TOISE will automatically optimise an interface to provide clues to the designer on how to place components on interfaces to alleviate some of the cognitive load and human motor functions when interacting with them, by using a cognitive architecture; namely ACT-R/PM. This approach is referred to in this thesis as an EvoCog approach. TOISE attempts to automatically evolve a near optimal layout for an interface, thus the essence of the research presented in this thesis is related to automatic optimisation of user interfaces. What TOISE will not do is to combine HCI heuristics or other design principles[1] (Lidwell et al. 2003) that designers may use as part of design and their usability activities. This toolkit was conceived to concentrate on human cognition and motor functions, as part of its functionality and its novelty is the combination of a genetic algorithm using ACT-R as a human simulator, replacing expensive and incremental human-driven activities with a relatively rapid and inexpensive meta-heuristic search of a much larger design space that normal human approaches would not be able to carry out.

## 1.1 Thesis organisation

The argument that is presented in this work is that a software prototype is developed by demonstration using the user interaction recorder i.e. Robot, which produces a complete and well-described interaction model for a specific interface. This model is then loaded into an evolutionary engine that uses some of its metrics i.e. locations of all the widgets contained in the interface and construct a population that is evolved using a cognitive model as part of its fitness assessment. TOISE will over time evolve an optimal

[1] Principles referred for instance to as 80/20 rule, accessibility, affordance, alignment, colour and consistency to name but a few presented in the Universal Principles of Design publication by the authors.

layout solution that will minimize both human cognitive load and motor functions when interacting with the interface for some series of specific goals.

The sections in this thesis are going to follow this structure. The first part of this document will concentrate on the introduction. Chapter 2 will provide a brief discussion on the historical background of HCI. Chapter 3 will review the literature related to cognitive modelling whilst chapter 4 will presents an overview of relevant research in the field of layout optimization using evolutionary algorithm. Chapter 5 will discuss in detail the TOISE approach and software, first by discussing the user interaction recording, then by presenting a comprehensive look at the cognitive modelling. The next section will present the evolutionary algorithms used in TOISE. Chapter 6 discusses a series of experiments demonstrating the abilities of TOISE to generate optimised design.

Finally chapter 7 will conclude the thesis by summarizing the advantages and the disadvantages of TOISE as well as problems encountered during the development of the software and provide some constructive directions for future work.

# Chapter 2. Review of major milestones and relevant literature

The evaluation of usability is increasingly becoming more important as a major part of any software development project and a range of methods and tools have been available to professionals and other groups to support their endeavour for some time. According to Andrew Dillon, usability is a measure relating to a user interface quality that is connected to its effectiveness, efficiency and user satisfaction when using this UI attempting to perform a series of tasks (Dillon 2005). This description of usability is close to the ISO 9241:11.

With the ubiquitous nature of computers (Christou 2007) and users experience and diversity becoming broader; software and user Interfaces must cater for this extensive range of users. Without a good knowledge of the target users, computer interfaces may become usable but their usefulness will often be debatable. Therefore, any computer system should be designed for the needs and capabilities of these target users for who there are intended.

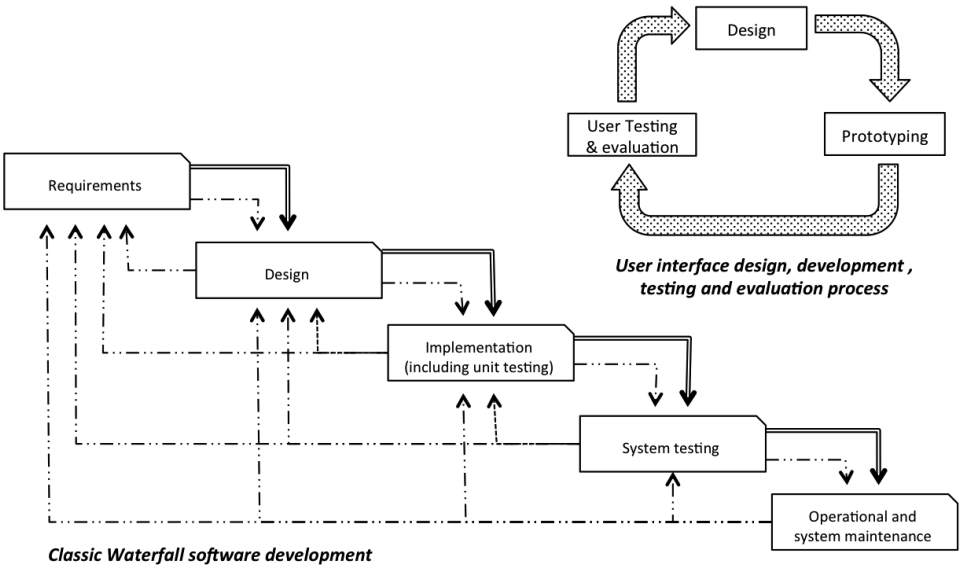



Figure 1: Waterfall software development process

One way to achieve this is by utilising a known approach that is gaining a wider acceptance (Bevan and Ferre 2010) named User-Centred Design (UCD), which includes the users throughout the design and development phases. This approach not only focuses on understanding target users but also focuses on understanding the tasks (or series of tasks) that are performable on the system, and furthermore the environment in which they operate in (Gersh et al. 2005). In addition, potential users are involved throughout

the whole development of the systems, thus insuring and acceptable level of usability feedback from one lifecycle stage to another. This approach contrasts with the more traditional sequentially based waterfall cycle (Stone et al. 2005) in which each part of the cycle is an independent part of the software development that is completed before the next phase in the sequence is started (Zachary et al. 2001). However in reality iterations between the different stages of development take place to allow some overlapping between the phases where information that is needed is fed back so alterations can be implemented (Zachary et al. 2001). The classic process is displayed in Figure 1. In this type of development, user testing and evaluation often take place at the end of the cycle which implies that by that stage changes are often more difficult and costly to achieve as the development is nearing completion, which also often precludes the information to flow from one stage to another as often one phase starts when the previous one was completed. This approach contrasts with the UCD approach (See Figure 2). As per the classical waterfall approach, it starts with a good knowledge of the target users' analysis. This leads to a set of requirements being drawn that contributes to the design of the software and its interfaces. Once a design is compiled, a prototype is created that is used as a basis for implementation (and user testing). The major difference between the UCD and the Classic approach is that users are involved throughout the whole process at each and every phase, and the feedback that is obtained is always taken into consideration at all stages of the development.

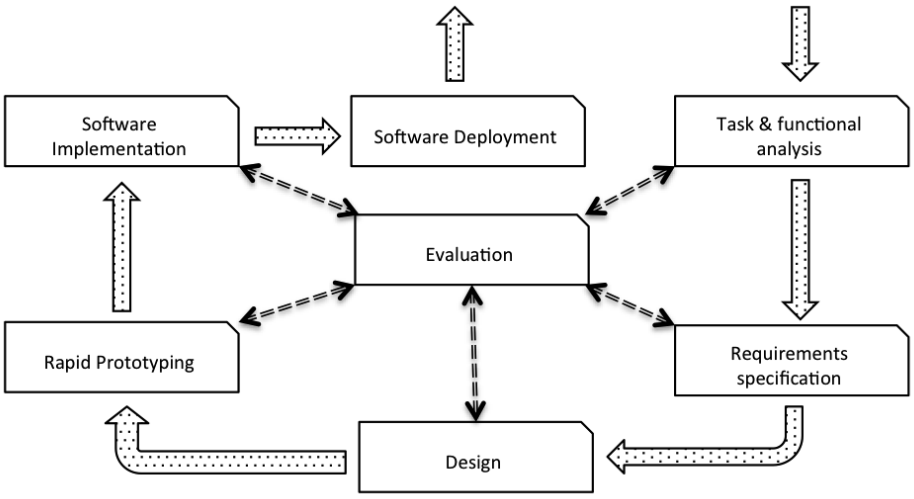



Figure 2: User Centred Design (UCD)

Although the concept of usability is perceived as key to the design of software and user interface in particular, its integration within the development lifecycle is often restricted to one user evaluation in more classical methodology (Hartwig et al. 2003; Kljajevic 2008). The UCD model of development tends in theory to re-adjust the potential of usability

evaluation by integrating the human-in-the-loop in all the phases of the interface development where the collected data can play a significant role to indicate usability improvements focusing on specific user groups for specific tasks. However, one of the main issues with usability evaluation is that it can become impractical or prohibitive in terms of cost i.e. conducting evaluation on population samples of pilots; doctors etc., but also in terms of logistics, that is consulting these participants repeatedly can be problematic (Byrne 2008).

One approach to solve this problem it the use of quantitatively realistic models based on these participants to replace the human-in-the-loop that can assess user interfaces, making the evaluation process more straightforward than using real users. These models, whether there are based on Goals Operators Methods Selections (GOMS) or cognitive architectures, act as surrogate users, which ultimately produce quantitative values such as execution times and error rates (Gamsriegler 2006).

The research presented in this thesis uses cognitive models as surrogate users and uses the metrics obtained during their usability assessment as the fundamental part of an optimization process, which ultimately provides user interfaces that minimize cognitive and motor load.

This essentially means that early in the design process, cognitive models are used to predict user performance when interacting with a specific UI architecture. This approach potentially allows developers to use evaluation data (provided by the cognitive model) which can help guide the design process and be of use well before any user evaluations are involved (Ritter et al. 2002; Fleetwood et al. 2006). It follows that predictions of human performance using software models has the potential to be of great practical value but, unfortunately, it is a difficult process to perform owing to the considerable difficulties with the creation of cognitive models based on and created from a set of criteria defined by experimental user test data. This is in part due to the complexities of collecting experimental data requiring specialist expertise and a great deal of resources. Unfortunately, there is not a great deal of data in the public domain, partly due to state-of-the-art research being funded by organisations such as NASA and the United States Air Force (Lebiere et al. 2005; Lane 2007).

The research reported and presented in this thesis involves an approach that will ultimately facilitate the modelling of human behaviour and the enhancement of user interfaces at the level of cognitive, perceptual, and motor operations using the unified

cognitive architecture referred to as Adaptive Control of Thought – Rational (ACT-R) (Sutherland 1964; Redmond and Smith 1975; Anderson 1996; Anderson et al. 2004; Lebiere 2006) which allows cognitive tasks to be modelled and thereafter run. ACT-R is discussed in detail in section 3.3.3. In short, ACT-R (Anderson & Lebiere 1998) is a modular architecture that embodies the theories of cognition, perception and motor behaviour (Das and Stuerzlinger 2007), which is composed of a series of interacting modules such as declarative and procedural memories, perceptual frameworks such as the visual and auditory modules and motor frameworks such as the motor and speech modules. They are coordinated by a central production system (Newell 1990), which matches the content of the different modules' buffer to fire the best production rule.

Thus in order to facilitate the modelling tasks to produce enhanced user interfaces, the system that is presented in this thesis involves the retrieval of cognitive predictions and metrics that will be used to improve the usability and design of these software interfaces in terms of efficiency in a real-time setting by using a genetic algorithm that will ultimately attempt to evolve a solution using the ACT-R predictions whilst a special module will retrieve all the user interactions relating to a specific prototype interface.

The main research question in this thesis is answered through two approaches: ACT-R and optimisation. This duality makes this work somewhat complex to present, as both approaches are based on fundamental principles i.e. human cognition and Darwinian principles.

For this reason, the following sections will present a historical overview of HCI and its emergence. Sections associated with cognitive architectures and modelling will follow where applications that relate to the presented research in this thesis are discussed.

Further sections on Genetic algorithms will explain some major concepts on evolutionary algorithms providing examples of usage. In the next Chapter TOISE is presented and discussed.

## 2.1 Historical background

There is little doubt that research during World War II led to new understandings of human factors (Kieras 1997) as new technologies were invented during that period such as Radio Detection and Ranging (RADAR) systems. Human beings became more active processors of information than they would have been before the development of these increasingly complex systems, e.g. in aviation that placed greater demands on human perception and cognition. This led to breakthroughs in experimental psychology, perception, performance, advances in our understanding of how human memory works, human learning processes and problem solving. It became apparent during this era that the main practical difficulties with equipment such as radar systems and military aircraft lay with the operating of these devices and not essentially with the complex devices themselves. In other words, the interaction with those devices was the major source of difficulty and not their complexity. This research i.e. Human Factors, influenced not only the design of those devices but the psychological theories that emerged during that time. In retrospect, this gave researchers the perfect applied problem to work with, and as society was being transformed due to the use of these new computational devices, cognitive scientists used their applied methodology to solve these problems that were critical to usability (Card et al. 1983).

With the emergence of early personal computers as opposed to mainframe computers normally used only by specialists, changes in human-computer interaction gave rise to new approaches and more understanding as cognitive scientists came to realise that with the new styles of interactions that were required using this new technology than better design of human-computer interfaces must become pivotal in dealing with human performance.

The first postulation was that the best approach to design should be based on task analysis, to obtain human interaction metrics and argued that since human activity is known to be goal orientated, and their limitation in terms of perceptual and information processing abilities was beginning to be understood, adjusting the task environment could increase human performance (Dorman and Gaudiano 1998; Bolton et al. 2012). Hence given the understanding of human cognition and motor functions and the goal to achieve, the task environment can provide information to psychologists to analyse how best this goal can be performed within this environment. Another postulation was the need to shift the methodology from results based on formal comparative work to results based on calculations i.e. approximations, which from then on became an essential tool to perform

their analyses (Card et al. 1980). To complete their set of requirements, cognitive psychologists who were becoming cognitive engineers argued one last requirement: a need for a supporting theory of human computer interaction relevant to the design of computer interfaces (Crystal and Ellington 2004; Grudin 2005).

It is during the design process, that the designer has still enough degrees of freedom to change the system usability as it is easier to propose evolved designs at this stage rather than when a system has already launched for use, which means that proposing any changes are extremely difficult to implement and would probably lead to a later major re-release. Therefore if psychologists only concentrate on evaluation of existing implemented systems, the impact of their research will remain negligible unless it moves in the design area (Carroll 1997).

Human Computer Interaction (HCI) emerged in the 1980s and has steadily expanded for over thirty years to attract many professionals from diverse disciples i.e. computer science, psychology, mathematics, engineering, merging concepts and approaches from those disciplines. HCI is an example on how many distinct approaches and conceptual interests can be integrated to provide a human-centred theory of best practice. It has evolved into a major area of research that concentrates on finding best practice with computer interfaces focussing on human interaction. However, as we will see later, the field of HCI and its focus has expanded dramatically over the past three decades to embrace new technological developments and the further diversification and complex user base.

In some ways, HCI was and still remain greatly influenced by technical advances. As equipment evolved, HCI had to catch up to fulfil its remit. This fundamental link between the two phenomena symbolizes two entities coevolving symbiotically.

The main technologies that affected the entire industry are both hardware and software advances. Without debating which one had the most impact, most agree that direct manipulation of objects, for example by utilising mouse pointer devices, created a shift in usability that has been difficult to surpass contemporarily. This however would not possible if the concept of Graphical User Interfaces (GUI) had not been previously invented. These two advances led to major adjustment of productivity and led to applications such as spreadsheets, word processing and graphic packages, which are now the rudimentary requirement in all computers. This drive towards a more interactive environment not only created an impact then but also paved the way for today's new

research on gesture's recognition and control, and many of the multimedia software used today, in their own ways, are revolutionizing today's industry.

In the next sections, we will discuss the main steps that evolved computer technology along with the many issues that the technological advances created and how the HCI community came about to solve the many problems the same technology brought about; in particular obtaining metrics or user interaction that this research is based on.

### 2.1.1 The first GUI and pointing device

The first computer graphics aided systems is often argued to have come alongside the first digital computers; the Whirlwind project being the first project to have used this technology which had over an acre of electronics with one control room containing one electronic screen display and a light gun providing direct input onto the screen (Machover 1994).

HCI would not be what it is without the phenomenon of direct input. The light gun was the first real direct input device created, used mainly in a military environment and was brought to light by the Whirlwind project[2].

Figure 3: Whirlwind Project (1949)

Created in 1944 as part of the US Navy's Airplane Stability and Control Analyser Project (ASCA), it attempted to provide a real-time flight simulator. The Whirlwind project was integrated into the Semi-Automatic Ground Environment (SAGE) that was an air defence system design to protect the US against enemy bomber attacks controlled by US Air Defence (See Figure 3). The light pen was created as part of the work on the Whirlwind project by Robert Everett. He designed this input device i.e. a light pen (or light gun) which when pointed at a light symbol displayed on a CRT radar screen (see Figure 4[3]). These two

[2] Project ran by the Massachusetts Institute of Technology (MIT)

[3] Courtesy of http://www.computermuseum.li/Testpage/Whirlwind-1949.htm

projects left a legacy by opening the door to the Computer Graphics (CG) discipline and showing CRT screens as viable displays and interaction interfaces, and moreover, introducing the light gun as an important input device for these devices (Redmond & Smith 1975; Lane 2007).

Figure 4: Sage System with early light gun

### 2.1.2 The next pointing device

The pointing device technology advanced when Ivan Sutherland (Sutherland 1964) invented Sketchpad[4] (See Figure 5[5]). Its concept influenced and galvanised generations to come as it fundamentally contributed to the domain of HCI, being one of the first usable graphical interface software.

Figure 5: Sketchpad (1963)

Even though the pointing device was not a mouse, the light pen used in his experiments allowed for direct input to take place. Using this device, objects could be drawn, selected and moved by dragging them onto a screen (Figure 6[6]).

[4] Sutherland invented Sketchpad as part of his PhD research in 1963-1964.

[5] Sketchpad: courtesy of http://kisd.de/~rbaehren/sketchpad.htm

[6] Sketchpad: courtesy of http://kisd.de/~rbaehren/sketchpad.htm

In the programmatic context, Sketchpad provided a major leap in techniques too as he abstracted what was displayed onto the screen in order to reduce cognitive load as his intentions were to create a system that could be used by non-programming individuals such as artists and engineering draughtsmen. In short his main task was to remove the division between users and programmers.

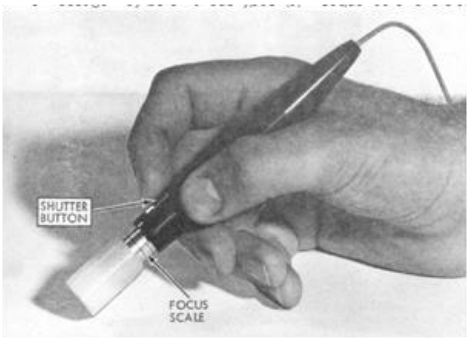


Figure 6: Sage System with early light gun

In that respect Sutherland succeeded and was one of the first person to use the concept of object orientation, namely the concept of classes and objects being instance of those classes with an introduction to inheritance. Many attributed Sketchpad as being the first CAD program for which Sutherland received the prestigious ACM Turing Award in 1988.

### 2.1.3 The ubiquitous direct input device: the mouse

Douglas Engelbart was a visionary[7] (Grosz 2005). His report "Augmenting Human Intellect: A Conceptual Framework" that described and discussed his vision and research agenda secured funding from ARPA, which led to the invention of an early prototype of the contemporary computer ubiquitous mouse[8] (Zhai 2004) (See Figure 8[9]). Originally made up of two geared wheels perpendicular with each other, the co-founder Bill English while working later at the Xerox Palo Research Park modified the original model and replaced the two wheels system with a metal ball pressed against some metallic rollers to translate the mouse movement See Figure 7.

[7] Engelbart earned a Ph.D. in EECS from UC Berkeley in 1955.

[8] Around 1963 at the Stanford Research Institute (SRI)

[9] Courtesy of http://www.dougengelbart.org/pubs/papers/scanned/Doug_Engelbart-AugmentingHumanIntellect.pdf

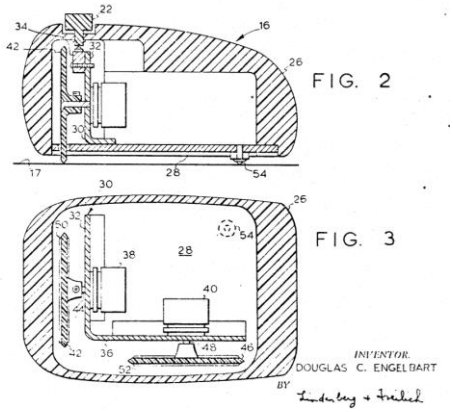



Figure 7: Engelbart's mouse patent: 3,541,541 - 1967

Some years later the mouse saw other modifications which included amongst others the additions of two buttons, to become the technology for direct input for their first personal computer containing a GUI as its interface. This computer was the Xerox Star as shown in Figure 11 [10].

Figure 8: Douglas Endelbart's first wooden mouse

### 2.1.4 Importance of Object Orientated Programming (OOP) in HCI

None of the advances made in pointing devices, which have become the main human interaction with computer user interfaces would have been possible without creating a layer of abstraction between the objects displayed onto a screen and their relevant virtual entities inside the programming code.

Nygaard and Dahl (1981) with their work on simulation models - SIMULA some years later would take the concept used by Sutherland much further and formalize what is known today as OOP (Nygaard & Dahl 1981). In short, most programming languages were based on functional decomposition techniques, i.e. based on sub-routines (or major actions) that were necessary to complete a specific task. With object-orientation, this

[10] Courtesy of http://images.yourdictionary.com/xerox

decomposition is based on objects. Thus changes in representation of an object are localized rather than global. Furthermore, the object-representation implements models of reality that are self-contained objects interacting and collaborating with each other hence introducing the idea of real-world objects for developing applications and systems (Booch 1986; Dahl 2004; Kindler 2007).

### 2.1.5 The importance of event handling i.e. Reaction Handlers in HCI

It is difficult today to conceive a programming environment without event handling which provides the main code response to human interaction on user interface widgets. However, in the early 1960s this concept was not formalized. There were influences such as the work done on the SAGE project, but this was a military project and consequently not in the public domain.

Another influence in this domain was the work of William Newman[11]. This researcher from Harvard University worked on the development of problem-orientated programming languages for graphical applications and Reaction Handler[12].

This fundamental concept lies on the fact that for every action taken by a user, a specific reaction would follow by the computer depending on the current state of the running program. This is undoubtedly the precursor of events handlers that is seen in most of today's programming software (Newman 1968). This paved the way for fundamental changes in ways software not only computes and retrieves data for specific tasks but also reacts to human interaction (Kay 1996).

### 2.1.6 Icons

In the field of HCI there have been many important scientific findings. Some were theoretical and pushed the science forward but some were practical and resulted in a major usability shift. David Canfield Smith's contribution refers to both. Heavily influenced by the invention of Sketchpad, and the new possibilities it presented, Smith went on to explicitly argue the benefits at the cognitive level of the abstraction between the user interaction and the programming platforms. In a sense it was clear to him that graphical images could represent abstract entities of some programming language objects.

Smith coined the term "icon"[13] to these graphical entities, a name still used today. These objects could be manipulated irrespective of what they were in terms of programming

[11] William Newman earned his PhD from the Imperial College in London.

[12] Newman presented a conference paper in 1968 based on the work done at Harvard.

objects. Smith being part of the team that later developed the Xerox Star as chief designer influenced the development with his principles and in particular with the first adoption of "WYSIWYG" (Smith 1975).

### 2.1.7 Xerox PARC

From this point, many computer scientists were moving towards the idea that computers that should not be solely for programmers and experts but also tools that anyone could utilise. This could have seemed an insurmountable challenge had the discoveries mentioned above not been made. The next stage to meet this challenge was to merge all these advances into one system. Alan Kay[14] was such a person to drive this concept to completion. In his book, Kay (Kay 1996; Chesbrough & Rosenbloom 2002) wrote:

> *"Ideally the personal computer will be designed in such a way that people of all ages and walks of life can mould and channel its power to their own needs."*

This typifies the vision that the Xerox team had in the mid-70s and led to the creation of the Dynabook in 1968 (See Figure 9[15]); a major evolutionary step analogous to today's tablet computers, directed towards usage by children (Kay 1972; Kay and Goldberg 1977).

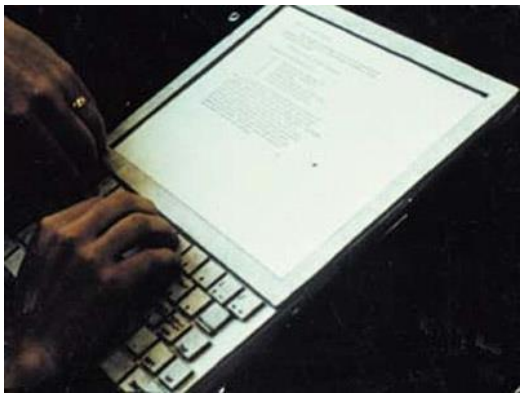


Figure 9: Dynabook (1968)

However it was later, at the beginning of the 1980s that Xerox developed their first "commercial" system to use direct manipulation extensively. Xerox first conceptual

---

[13] Based from Smith's PhD research on Pygmalion in 1977,

[14] Working at Xerox PARC

[15] Courtesy of http://history-computer.com/ModernComputer/Personal/Dynabook.html

system, was the Xerox Alto in 1973[16]. It was a device used only for experimental purposes and incorporating the new innovations that were made at the time.

Those computers mainly used in Xerox's original Palo Alto Research Centre (PARC) (Irby et al. 1977) were revolutionary[17] in a way that they were Ethernet-linked and provided unique services (for that point in time) including arguably the first Graphical User Interface (GUI) (See Figure 10[18]). Later, the Xerox Star development started in 1977 (Smith et al. 1986). The mission behind this development was to create the "office of the future" and owed most of his technical success to the Xerox Alto[19], which incorporated many developers from the Alto project (Chesbrough and Rosenbloom 2002).

Figure 10: Xerox Alto (1973)

Despite the huge technical breakthroughs this computer brought to the world, the Xerox Star (See Figure 11) failed commercially in 1981. However many agree that this engineering achievement laid the foundation for a new type of computer interfaces based on “WYSIWYG”, Ethernet, network file services and printing services which are still with us today.

[16] Development team was headed by Chuck Thacker.

[17] Thacker was awarded the Turing Award for his pioneering work with the Alto

[18] Courtesy of http://retro-treasures.blogspot.co.uk/2010/10/xerox-alto-computer.html

[19] David Liddle was in charge of the development team.

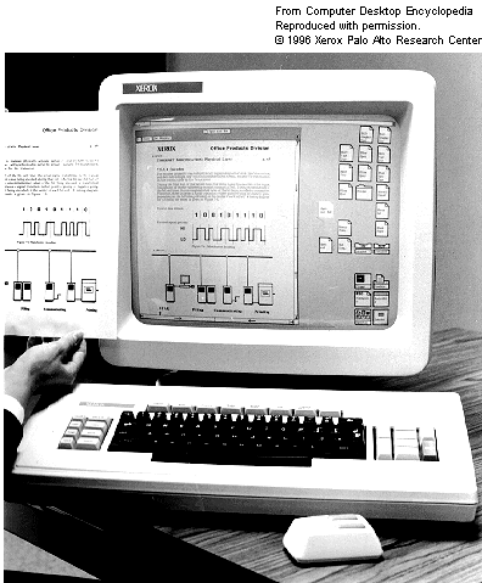



Figure 11: Xerox Start personal computer with a two buttons mouse

### 2.1.8 Apple Macintosh

The Xerox Alto did not just spawn interests within the Xerox team. In the late 70s, the Apple management became aware of the revolutionary new architecture of the Alto and decided to use its new mouse driven GUI concept for a new system they were designing to upgrade their existing commercially successful Apple II. The Apple Lisa[20] (Figure 12) presented a GUI not unlike the current Apple "Look and Feel" was introduced in 1983.

Figure 12: Apple Lisa

Three years later, in 1986, the Macintosh Plus was introduced including some fascinating improvements to earlier models on both the hardware and applications fronts. One of the major enhancements this product contained was the inclusion of a SCSI port, which allowed many types of devices to be attached to the system externally including

[20] At the time Xerox was a shareholder in Apple the Apple Lisa.

extra display units. This “WYSIWYG” system included many programs such as MacPaint (Figure 13 [21]) and MacWrite (Figure 14 [22]) bundled into it. In addition, third party software like Word and Excel [23] and PowerPoint[24] could optionally be installed.

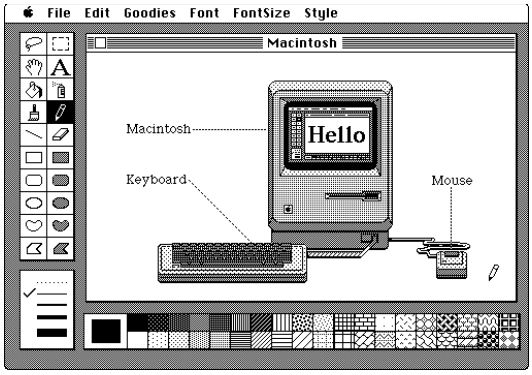



Figure 13: Apple MacPaint

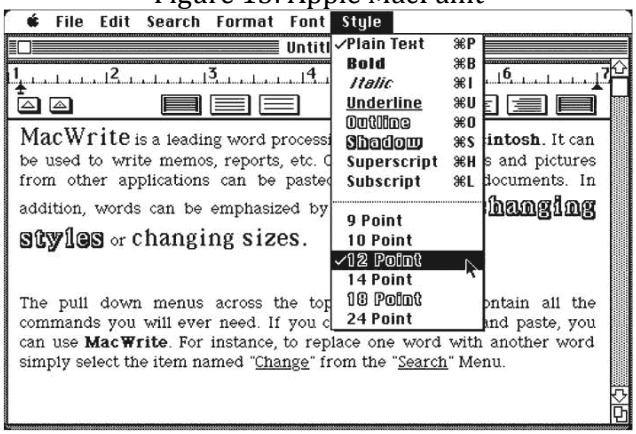



Figure 14: Apple MacWrite

All of these technological breakthroughs have had a major impact on the way people interacted with computer systems. Allowing human interaction with computer interfaces created a breeding ground for research in usability as these user interfaces became more complex and failures in this field had major impact at many levels both in terms of productivity and cost. What was true yesterday is still true today and as computer interfaces become ever more complex and human limitations on system performance is being ascertained, usability evaluation is now beginning to be taken seriously. With the increasing cost of user evaluation due to the increasing popularity in approaches such as the UCD approach, user modelling is becoming more popular as a mean of evaluating user interfaces. Tools have been created to help designers with user interface evaluation, but

[21] Courtesy of http://churchm.ag/macpaint-and-quickdraw-source-code-available-mac-download/

[22] Courtesy of http://www.jnkmail.com/stuff/nostalgia/

[23] Word and Excel where first produced by Microsoft for the Macintosh platforms.

[24] PowerPoint was created by Forethought before being bought by Microsoft.

the system presented in this thesis goes one step further and provides a mean of automatic optimisation of those interfaces.

## 2.2 Software Evolution

The field of text editing had the major impact on the whole software industry. The first prototype came from the Stanford Research Laboratory and was proposed by Engelbart[25] where he implemented the first word processor in 1965 called oN-Line System (NLS) after three years of research and demonstrated it in 1968.

NLS marked an evolutionary turn in computer office software. Engelbart who also invented the mouse was a pioneer in human-computer interaction research and his team put forward a new concept of hypertext. This word-processor had similar functionality to the bestselling software that came out some years later: WordStar. It provided a turning point in the way text editing would be considered and marked the start of major funding in R & D that would ultimately lead to word-processing software such as WordStar leading to MacWrite and Microsoft Work.

Graphical programs are a major part of software tools used today. In the early days, Sutherland's Sketchpad using a light pen first introduced this technology. The real demonstration of real time user interaction with a mouse came with Engelbart's NLS. With the continuing research drive that was taking place, another significant step in the evolution in human computer interaction was taken in 1968. Ken Pulfer and Grant Bechthold[26] using Engelbart's idea of a device pointer drew all the frames of a movie sequence using a key-frame animation system. This led the director Peter Foldes in 1971, to direct the first computer-animated movie produced by the National Film Board of Canada (See Figure 15). This film became a success and was recognised by the awards bodies in and around 1971 i.e. Jury Prize at the Cannes Film Festival, a BAFTA award for Best Animation Film and a Silver Higo at the Chicago International Film Festival. This iconic film had a major impact in a somewhat traditional movie industry.

[25] Engelbart was a researcher at the Augmentation Research Centre (ARC) at the Stanford Research Institute (SRI) during the 1960s.

[26] From the National Research Council of Canada

Figure 15: Hunger by Peter Foldes (1971)

During the same period other influential research took place for example, Newman when working on the Xerox Alto project implemented the first paint program. Soon after, Patrick Baudelaire's Draw became part of the Alto successor - the Xerox Star system. The first real Paint program is agreed to be Dick Shoup's SuperPaint[27]. This pioneering research and development from Xerox PARC with colleague Smith increased the dynamics towards a technology that would revolutionize the multi-media industry in terms of features. Not only, would the program allow the capture of images from standard video output, but also virtual paintbrushes, pencils and fills. Moreover, SuperPaint became the first Paint program to use and depend entirely on a GUI with the new feature called anti-aliasing using a pointing device. Television channels used this relatively easy to use system to create custom graphics for their programs. NASA for the Pioneer Venus project mission in 1978 used SuperPaint to create their technical graphics and animations. Shoup won an Emmy award in 1983 for his work, and an Academy Award shared with Smith and Thomas Porter in 1998, for his development of SuperPaint (Shoup 2001). This bespoke system was never intended to be for mass production but like Newman's Sketchpad and Engelbart's mouse and his first word-processor, it became part of a surge of new ideas and concepts that would revolutionize the world of HCI.

## 2.3 The emergence of cognitive engineering

With windows in a Graphical User Interface (GUI), with pointing devices and concepts that allowed developments of many of today's software and hardware devices, the world of computing and computer users changed beyond recognition. This revolution in terms of user demands will put pressure on the HCI and Cognitive Science communities.

[27] Shoup earned a PhD from Carnegie Mellon University in 1970.

Prior to the 1970s, technology professionals were most likely to interact with computing technology. However the emergence of personal computers in the 1980s revolutionized not only productivity in people's professional lives but also at a private level. People not only used computers at work, but also started to spend time at home utilizing them for recreation purposes, such as game playing, improving their personal lives using word-processors and interactive programs such as communication software. This led to an explosive growth of computer usage, which changed the fabric of our society and human behaviour.

This surge highlighted many deficiencies of computer devices as pieces of hardware as well as in their interfaces with respect to usability for the individuals wanting to use computers as tools. It also presented a major challenge to the manufacturers, designers and programmers of those artefacts who after being made aware of these deficiencies who lacked formal theories and models that could guide design as well as methods, tools and techniques to build usable systems.

Fortunately, this corresponded to the time when cognitive science, which included domains like artificial intelligence, psychology, and linguistics, emerged. The professionals from these fields became active, began to work together, published their findings and with the help of manufacturers made sense of increasingly complex technologies and the changing of user populations which led to the emergence of modern HCI (Roth et al. 2002).

The main core of the research started yet again when Xerox, established the Palo Alto Research Centre (PARC) in the 70s, which was by that time famous for its developments in interactive technology. In those days, Allan Newell, employed as a consultant to undertake some psychological research into human interaction using the newly invented technological concepts such as user interface and interaction devices, created with his former students Stuart Card and Thomas Moran a small unit called the Applied Information-Processing Psychology (AIP) project with the main remit of applying their applied psychology research to HCI (Laird and Rosenbloom 1992). In short, their work concentrated on the quality of human interaction when using interactive devices such as mouse and GUI. Their efforts amongst others placed HCI at the centre of one of the most influential place in terms of new technology. From this research, a very influential document emerged written by Stuart Card, Thomas Moran and Allan Newell: The Psychology of Human-Computer Interaction, which introduces the concepts of human performance, GOMS and KML. The latter two concepts being paradigms that allows the

simulation of users, will be introduced in the proceeding sections; concepts still in use today.

The main remit of this new venture, was to include systematic and scientifically understood concepts into applications development and assessment focusing solely on usability based by research made by cognitive scientists. It gave rise to "Cognitive Engineering". Cognitive Engineering is an interdisciplinary approach to the development of principles, methods, tools, and techniques to guide the design of computerized systems intended to support human performance (Woods and Roth 1988; Roth et al. 2002). This emergence came at the right time and injected new and novel concepts, integrating skills from different disciplines, and most of all delivered a new needed vision dedicated to addressing deficiencies with the use of computers and digital devices.

Early research focused mainly on software such as text input and editing as well as data entry in software. Today, it has become far richer and has expanded to areas like information systems, visualization and many aspect of product design to name but a few. Thus, in the academic world, HCI is taught in departments such as psychology, design, product design, cognitive science, science and technology, it is also included in industrial and manufacturing courses. HCI is now an integrated part of every taught concept integrating different approaches, and this richness encompasses a broad spectrum of users ranging from the young to the elderly, users with many forms of disabilities from learning to physical disability ranging from motor to visual impairment. As software domains became richer in terms of functionality, HCI focused on many aspects of software (and hardware) applications and now includes games, e-software, military software and process controls. This spectrum of applications meant that the early primary focus of HCI which mainly dealt with graphical user interfaces and usability now includes many other types of interactions, for many types of devices, ranging from hardware devices of any sizes i.e. from computers, hand-held devices, and mobile devices of many types with all the intricacies that these new devices offer.

This broad landscape is very varied and complex and develops professionals that once would have been known as computer scientists and psychologists, but now produces user experience designers, application designers, user interface designers, interaction designers and other relevant engineers for these professions. The application areas are now sub-divided into many domains. These subareas include mobile computing, in-vehicles systems, handheld and wearable devices, distributed systems, programming tools and their related tools.

## 2.4 HCI focus

The main focus of research in Cognitive Science (CS) is to study and to develop mental models as part of its efforts to understand and predict how humans know, perceive, make decisions, and construct behaviour in different environments.

Cognitive Science is the study of how the human mind operates and has inherited many of its concepts from psychology, itself associated with the study of human behaviour. Cognitive Science focuses on the understanding of how the human mind works not only by studying human behaviour but by incorporating research from many other fields not limited to linguistics, neuroscience, artificial intelligence and psychology. Cognitive scientists have many tools at their disposal i.e. for example computing and mathematical science, to represent some of the inner-workings of the mind such as thinking or addressing several processes related to the working of the mind like acquiring or using knowledge. One of the main contributions of Cognitive Science that has much weight and heavy impact on the research presented in this thesis is the development of theories, which led to the creation of cognitive architectures that account for human cognition and behaviour.

HCI adopted and adapted these concepts to further the study in its main area of research i.e. usability. This approach is at the centre of its success. For instance, HCI applies Cognitive Science concepts to deal with information visualization in areas such as user modelling. The counterpart is seen when HCI provided cognitive scientists the guidance in regard to its user research (Carroll 1988).

The collaboration between HCI and Cognitive Science, which is far-reaching, led to theories like models of human information processing that were frameworks used to explain and describe human mental processes, including basic aspects of perception, attention, short-term memory operations, motor functions integrated into a single powerful model. Two implementations of those concepts are GOMS and KLM (an example of GOMS model is given in Appendix A) which will be discussed in later chapters (Card et al. 1983). Because of the dynamic aspect of HCI, these basic functions were developed further. From an atomic actor, which was what the early model described, in the late 1980s and early 1990s, the HCI concepts have now evolved to integrate more fundamental concepts which led to the integration of the Activity Theory, whereby a user is not seen at an atomic level separated from his/her environment, but rather part of a collective framework. These conscious activities with objects can be broken down into goal-directed

actions through predefined automatic operations that have to be performed in order to satisfy these objects. The complexity is that these activities are not fixed but can change dynamically as the environment or conditions to which actions are applied, are dynamic and changes.

## 2.5 Design deficiencies

Given the impact that HCI has had on the industry, deficiencies in software and hardware devices are still commonplace. The main question is why this state of affair still exists today, given the massive development drive and theoretical impact HCI provides to the industry? To answer this question, we must first discuss what might be referred to as "mainstream user interface design". The step often mentioned is the stage where one does try to define what the software / system does and who is going to use the system. The stage is referred to as tasks and user analysis. Not only should the system be able to achieve what it is intended for, but also integrate well with the world of users.

Traditionally, a requirement analysis is performed on a system. This leads to a set of specifications that is used for the development of the system. However, this methodology is not user centred, but is system centred based on assumptions of usability for a particular system by the system designers. If one looks into the details of what tasks a user must perform to achieve a series of goals and evolve a design accordingly, this puts the user back at the centre of the development.

One of the major obstacles if a development team decides to undertake a thorough user analysis is often not the process to render the system more user-centred itself but to recommend this need to their management team who often have financial and timing pressures. In addition, this often involves additional professionals and many testing users, pushing the cost of development. Such usability evaluation activities actions are often rejected on ground of extra cost even if it is common knowledge that an early and continued involvement leads to good design and therefore gives a more likelihood of the software being successful (Karat 1993). Alternatively, developers (often prescribed by early management requirements) might use a "template" of what the system may look like (Apple 2012; Microsoft 2012). Creating a new kind of interface often results in failure and management team being aware of this might set a basic set of specifications long before development is started. The beneficial points in following this road can have positive outcomes. First this road minimizes the uptake risks, as most users will be familiarized with a known interface template. Secondly, the design cost is minimal, as most of the

design research has already been done. This type of project often relies on backend design and development. A very good example of such templates is with Windows and Apple's interfaces. Not trying to compare them, one can see that using a formal interface template for designing a User Interface (UI) with those platforms will ensure user satisfaction both in terms of use and aesthetics. A user familiarized with a type of template interface for instance will almost be competent using a system that contains interfaces similar to those already used.

Good practice should be at the centre of development and should cover a good task analysis, which should itself cover a detailed use of the system through series of real tasks. Once this information is collected from human user testing, it can be used to analyse the existing system and draw another set of specifications where the users are at the centre of the changes. The tasks included should attempt to capture most of the software difficulties in terms of usability. The new specifications are then used to optimize the software. The momentum is kept well into the "alpha" phase of the development, thus allowing the maximization of usability for a particular system. This objective can only be kept by investments both in terms of financial support but also in terms of dedication of the development team both at a managerial and development level. The cycle just described needs a lot of user testing resulting in expensive extra costs and additional time allocation. Moreover, it will often impact on the development cycle, which could stall while the usability testing is undergoing its course. Hence, to factorize all these aspects into a system development makes prediction difficult and more complex to formalize in terms of milestones for instance.

Many important principles of design were proposed by Carroll and Rosson (1997) in the form of the Scenario Based Design (SBD) approach. SBD inherits the components of UCD (see Star model in Figure 2), but adds another dimension by including the concepts of scenarios in the development cycle. Scenarios are stories about people behaviour, their goals and the environment they are. They have a plot i.e. they include actions and events that either happen to actors or are generated by actors (Carroll 1999). They can be written from many perspectives and for many situations, and for many purposes (Carroll 2002). They are techniques that are used to shift the focus of the design process from mainly purely functional specifications to a description of on how people use a system to achieve a specific task in a specific environment. In other words they are a kind of lightweight user interaction visualisation of the use of a system by potential real target users that have become popular as they enable a realistic and rapid communications about potential usage of a system amongst many parties in the development cycle (Rosson and

Carroll 2009). Furthermore, in terms of user analysis, SBD tends to supply important information that is useful to include cognitive engineering into the loop.

Perusing Figure 16, one realises that the user is a central part of the development of the system. From the user analysis, much information can be conveyed to form scenarios and scenarios can be used to help researching human information processing specifics. Scenarios provide important information to the user centred design process to direct the design process. User evaluation takes place to refine requirements, which can lead to new scenarios being created.

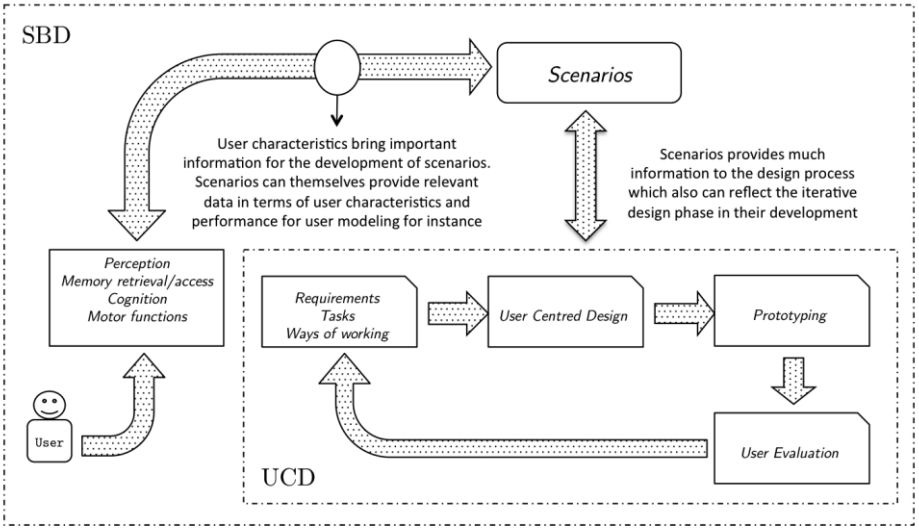



Figure 16: Scenario Based Development

This forms the basis of SBD, which is at the centre of many development projects. The main issues with empirical testing are its time scale i.e. slow approach and financial pressure put on software companies i.e. can be prohibitively expensive if domain experts are involved, to apply an appropriate usability testing program (Kieras 2006). For example a Norwegian company called TNO (TNO 2012) can charge anything between 50,000 to 100,000 Euros, with tests lasting between an estimated 8 to 18 weeks to perform some usability testing on a company behalf. These figures represent a typical example of the bottleneck for the adoption of usability studies into the development of software discussed above. Moreover this is compelled by the fact that user performance varies greatly over a set of experiments and participants (Ritter et al. 2002). There are approaches such as usability discount cost benefit methods that help with those issues (Rajanen and Jokela 2007; Rajanen 2011). However another alternative came with the cognitive engineering community response by creating cognitive models specially engineered for usability, providing quantitative predictions such as task completion latency describing how well humans perform a task or a series of tasks using a specific

design; models that can be used as surrogate users to get empirical data (Kieras 1999). Such an approach provides an opportunity to iterate through many design revisions under a variety of condition much faster than real time (Byrne 2003). The research this thesis presents relies on surrogate users that have been modelled and engineered to run using a cognitive architecture, which will be described in later chapters.

The synergy between Cognitive Science and HCI helped to provide some important developments that are far more reaching than the main usability focus. We have mentioned above the dynamic aspect of HCI evolution as systems complexity increased and ways it moved on to encompass this complexity in its research. Cognitive Science followed the same evolutionary path in its own domain by developing more advanced human models and architectures capturing not only based on basic human reactions and motor functions but also fully functional models born from research in psychology and neuroscience. Thus a new era in Cognitive Science grew alongside HCI, which had a major impact in our understanding of human interactions with the external world. This synergy brought about theoretically speaking a huge amount of research that helped all professionals involved in developing hardware and software devices to understand usability for ever changing and involving user needs at a more meaningful way in the development of systems. The advances in human modelling gave HCI the tools needed to perform usability tests without human user involvement.

These tools are discussed in more detail in the following chapter, starting with the Model Human Processor (MHP). MHP was the first real attempt to formalize human behaviour and interaction. The extensive research gave birth to human interaction modelling that led to the creation of models such as Goals Operators Methods Selections (GOMS) and later Key Stroke-Level Model (KLM) (a subset of GOMS focusing on computer interface interaction) both of which are explored in the following section. There were followed by the advent of cognitive architectures. In the next chapter, an overview is given, concentrated on aspects that have an impact on the research presented in this thesis.

# Chapter 3. Cognitive Modelling

## 3.1 Model Human Processor (MHP)

The book *The Psychology of Human-Computer Interactions* from Card, Moran and Newell[28] (Card et al. 1983) is often credited with bringing to the forefront the concepts of Human Computer Interaction and human cognition and is thought of as one of the most central early influences in human computer interaction and cognitive engineering-oriented usability fields. It introduced MHP as an engineering model of human performance and GOMS as a methodology to perform task analysis (Byrne 2001).

This section provides an overview of MHP and is heavily based on the early publication of the book mentioned above.

The importance of MHP will be illustrated using this publication as a reference, why it is considered as a turning point in the impact HCI has had in relation to new developments in computing science in general, in which applied psychology comes to the rescue of developers and engineers to describe and explain how humans interact with computers devices.

A simple model of human performance was created which gives an integrated description of psychological knowledge about human performance relevant to human-computer interaction. It is called MHP. This model sees the human mind as an information-processing system that can be used to predict and approximate user-computer interaction (see Figure 17).

The MHP is an elementary model of human performance, intended to equip cognitive engineers with rough estimates of system behaviour focusing on the information processing capabilities of an average individual, described in terms of sets of memories and processors, in addition to sets of principles connecting human operations to these sets.

In terms of memories, the MHP includes sensory memories which are small buffers that hold information sensed from the environment before it is eventually encoded, short-term memory (or working memory) and long-term memory which are buffers that hold symbolically encoded information.

[28] Published in 1986

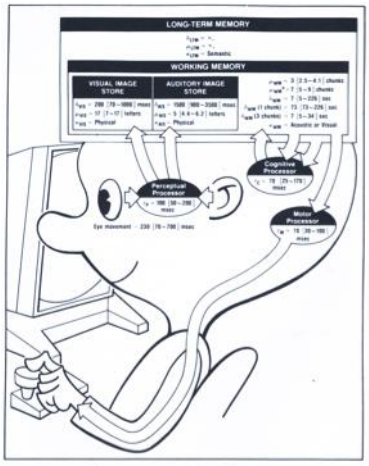



Figure 17: The Model Human Processor

The processors in this model refer to systems that can act on the sensed data. There are three types of processors referring to the *perceptual system* that includes all the human sensors and their associated memory buffers, the *cognitive system* that receives symbolically encoded data from the sensory stores and its Working Memory (WM) and responds using previously stored long-term memories and finally the *motor system* whose main task is to respond to the cognitive system.

The Perceptual System (PS) transfers sensory data of the physical world that is detected by the human body sensory systems into symbolic representations. The data received by the human sensors is coded physically supporting physical properties such as width, height, curvature, and intensity for visual stimuli or pitch and intensity for auditory stimuli. In this model, processor memories are associated with working memory. Another aspect of this memory model is that this information decays over time. This is called the Decay Time (DT). MPH measures this value in terms of half-life values. The perceptual processor, which acts on perceptual memories, is the time taken for the processor to respond to a given stimulus.

The Motor System (MS) can be thought of a system that translates thoughts into movements and plans patterns of muscle activations. HCI is mainly concerned with effectors such as arm-hand-finger system and the head-eye system. Like most of these motor activities, the movements are not continuous but consist of series of small micro-movements or motor saccades. Like many of those values, this value is an approximation based on research results. When dealing with the motor system, it is difficult to exclude the feedback system that exists between perception and motor functions. For example, if one wants to complete a motor movement with some precision, one needs to make some small corrective adjustments in order to complete the task. Therefore one intended motor

movement latency is often a summation of all the timings obtained for different sub-action that led the full action to be completed.

The cognitive system is involved in passing stimuli from the perceptual system responses to the right motor output system. Two types of memories are associated with the cognitive system: the working memory or short-term memory, which holds the current symbolic information, and the long-term memory whose main purpose, is to store this information for future use. All the information that is generated by the perceptual system is stored in the forms of symbols in short-term memory. Hence all mental operations will refer to this memory to gain access to the current registers used by the different perceptual systems.

The abstract symbols that are stored in this memory are called chunks and are usually nested. This means that a chunk can refer to or include another, the latter referring to another one etc. These relations are therefore useful for memory retrieval purposes as these relations mean that when the information stored in a chunk is activated in the long-term memory, this activation will spread amongst related chunks and thus makes those chunks themselves more retrievable by virtue of increase activation. It will also affect the retrieval speed as more activation a chunk has faster retrieval time. However due to human limitations, if those chunks get more activation, some others will see their activation decreased in time if not reactivated due to decaying itself proportional to the number to chunks. Like any other kind of memory, working memory decays too. This decaying time was researched quite extensively to show the relevance of the number of chunks to the decaying time and demonstrated the human limitation when recalling information. For instance, when a person is asked to recall some information that they just heard a few seconds before, both working and long-term memory are used in this process. Results showed that the storage capacity of the working memory is between 2 to 4 chunks. However if given a little more time is given (>5 sec), the long-term memory is also used in addition to the working memory and the number of chunks that one can recall increases to 5 to 9 chunks. This number of chunks is well known and familiar and demonstrates the limitations of the human cognitive system as well as re-enforcing the proverbial sentence: “take your time to think about it”.

The knowledge stored in the long-term memory can only be accessed through the working memory i.e. it is transferred from long-term memory to working-term memory. It is common knowledge that this memory acts as a massive storage space knowledge that is gained over one’s life. It is understood that long-term memory does not decay over time.

The major point to of discussion regarding the links between chunks in long-term memory is that most of the retrieval process depends on those links or semantic coding that exists between them. The situation in which the encoding context happens is important for retrieval purposes and leads to the difference between accessible and available memories. Whilst the latter encompasses all of the memories that have been encoded, the former deals with memories whose symbolic encoding has been achieved in the same context as the cues given for their retrieval. This is called "Encoding Specificity". This principle states that the likelihood of successful retrieval increases as the similarity between retrieval context and the encoding context increases. In other words, to maximize the retrieval rate, the same context should be referred to when memories were encoded during the retrieval process. It also follows from the above, that given a particular chunk (or set of chunks), the more associations it (or they) may have then the greater it will be/their chance(s) of retrieval. This is a very important factor in terms of retrieval performance. In terms of cognition, the optimum strategy to ensure easy retrieval is to attempt to associate a symbol with others already encoded in long-term memory in some ways and furthermore, to attempt to do this in a novel way, to avoid interference with other piece of information that may lead to retrieval failure. Finally, following Card's MHP model, the information get stored in long-term memory once the decay time threshold of the working memory i.e. around 7 sec, is passed, which compared to the time of retrieval i.e. around 70 milliseconds is far higher. This implies that if the task's latencies are very small then, storage in long-term memory cannot occur.

The cognitive processor that Card describes is similar to the computer CPU's fetch and execute processes i.e. the information is fetched from some registers (using different registers depending on the task to achieve) and then processed which may lead to some registers having different data presented or stored within them; the CPU clock controlling the flow of executions. The same applies in cognitive processing. First the content of working memory is pattern matched with the long-term memory content. Second, the procedural content will be executed which might lead to the working memory content to be updated with new chunks of information, and the cycle repeating itself. An important point is that this nominal time will decreases with practice or the demands for it.

One interesting detail about the cognitive processor in this model is that the recognising processes i.e. pattern matching it spawns run in parallel but the processor schedules all the necessary procedures in a serial manner. This bottleneck has serious consequences: the human cognitive system can be aware of many things at the same time through its perceptual and motor systems but it will only be able to spawn one action at a

time to deal with this new acquired awareness. In addition, as this awareness depends on the perceptual and motor systems, which run in parallel to the cognitive system, one can be aware of many factors that may happen around one-self but one will ever be able to deal with one phenomenon at a time. Card's & al MHP had major consequences for the future of usability and HCI overall and led to the definition of GOMS discussed in the following section, a framework for task analysis and later its simpler version called KLM; both of which allowing for empirical predictions to be generated for those models. It paved the road for production-system representation of human procedural knowledge used by much cognitive architecture today.

MHP was a major achievement in the 80's, which embedded psychological human behaviour findings known at the time. However, today, it is out-dated, lacking most of the recent finding in research in fields such as neuroscience. Major issues with MHP was its manual processing and its reporting of only expert behaviour. A more useful model stems from this research i.e. GOMS, which became a focus in HCI and is discussed in detail in the next section.

## 3.2 Goals Operators Methods Selections (GOMS)

The Concept of the MHP representation helped to define a general model of human interaction. Research by Kieras and many other researchers demonstrated how GOMS analysis (based on MHP) could be used to obtain usefully accurate predictions of learning and execution time dealing with computer interaction.

GOMS stands for Goals Operators Methods and Selection (GOMS). The Goals represents what the user has to accomplish using the software. It can refer to one particular task or be broken down into series of sub-tasks; all of which are to be completed for the overall goal to be achieved. Operators are the elementary or complex motor or cognitive actions that must be performed to complete a task. They can change the user internal mental state as an action is taken place but they also can change the physical environment around that user.

Since MHP, GOMS high-level predictive modelling has been accepted as one of the most widely known and used theoretical concepts in HCI which evolved to several variants, some of which are executable (John and Kieras 1996). In terms of modelling, GOMS is perceived as a high-level language in which user interaction is expressed in a hierarchical manner that represent a decomposition of complex tasks into more elementary ones (St-Amant et al. 2006).

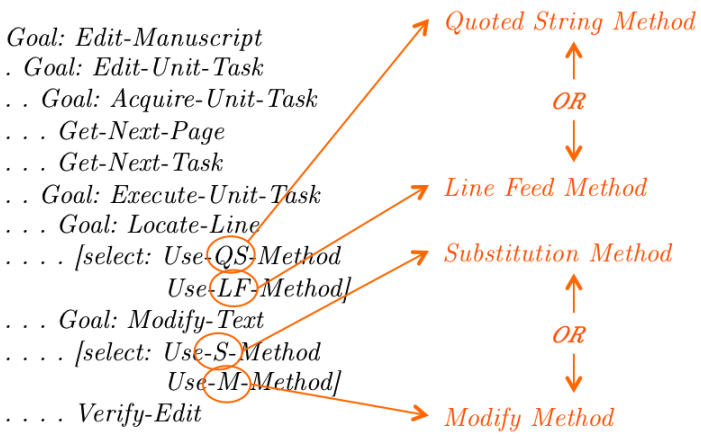



Figure 18: Outline of the GOMS model for a line editor (Card, Moran, & Newell, 1983)

GOMS models (See Figure 18) are descriptions of the knowledge that a user must possess in order to perform some specific required tasks on some device. It is the procedural knowledge that is required in order to accomplish a task (Kieras 1999).

There is an important factor about Operators, which is the execution time often approximated by a constant discovered empirically. In simple terms, Operators are the actions that the software allows the user to achieve. Methods are the sequence of Operators or sub-goals that are needed to complete some goal that depends on the nature of the specific task to accomplish. In short, they are learned sequences of operators and sub-goals that can lead to the completion of a goal and can be perceived as strategies to accomplish a task. One can have many strategies to complete similar tasks but only one of which is selected. Selections rules are the learned decisions procedures that must be used in a specific context to select the required method. It is the case that a user may have learned different methods to complete the same task for instance. A method of selection is necessary depending on the context to decide which strategy is better suited than another. In a GOMS model, a Selection Rule would exist to select the best strategy i.e. method to carry on a series of tasks to complete a goal (John & Kieras 1994).

There are many flavours of GOMS ranging from simple to more complex forms. One of the simplest is KLM (Teo and John 2006) that was first presented as a concept by Card and his colleagues. This version focuses on human physical actions adapted to computer user interfaces. The model includes a sequence of simple motor or mental operators such as *click-mouse*, *type a letter* or *thinks* to perform a required task. KLM provides focused time latencies predictions that are required to perform a task.

Others more complex variants are Cognitive Perceptual Motor GOMS (CPM-GOMS), Natural GOMS Language (NGOMSL) (Gray et al. 1993; John and Kieras 1996; Kieras 1996; Kieras 2006).

GOMS, KLM and other versions mentioned in this section suffer the same issues discussed for MPH. However the more complex versions bring extra functionality that has been found very useful in HCI (Gray and Altmann 1999). For instance, CPM-GOMS provides means to chart dependencies between the user's perceptual, cognitive and motor processes but is still an approach that models skilled behaviour of users (Biswas and Robinson 2008). However in terms of human performance, despite the fact that CPM-GOMS allowed parallel execution of user processes, resulting in better predictions for skilled human behaviour, its predictions are only based on estimates of latencies for primitive human cognitive, perceptual and motor processes (Vera et al. 2005).

## 3.3 Overview of Cognitive architectures

A cognitive architecture, whose concept was first proposed by Newell (Newell 1990) in an attempt to challenge the research community to develop a Unified Theory of Cognition (UTC) that simultaneously address all aspects of human cognition and is a framework that simulates the fixed mechanisms of human cognition (Kljajevic 2008). In so doing, a well-defined cognitive model can be developed specifically and executed by this architecture. Newell first proposed a production system, which became the basis for much of the cognitive architectures that exist today, and went on to develop with Laird and Rosenbloom the SOAR architecture. Later ACT-R was developed by Anderson (Anderson and Lebiere 1998) and EPIC by Meyer and Kieras (Kieras and Meyer 1994); the latter being the first to include perceptual and motor mechanisms, each representing a synthesis of recent empirical evidence gathered at the time, thus addressing the full spectrum of behaviour; from perception, cognition, to motor functions (Chong and Wray 2006). ACT-R was extended to incorporate perceptual and motor processes, resulting in ACT-R/PM (Byrne and Anderson 2000).

Cognitive models based on cognitive architectures have been used in human-computer interaction in three ways: from prediction of task performance times and errors (Van Rooy et al. 2002) to assist users in tasks performance (Ritter 2003) or to be used efficiently as substitute users during testing (John and Prevas 2003).

The work presented in this thesis makes extensive use of cognitive models that are run to gain metric of human performance. We will present a summary of three most used

cognitive architectures i.e. SOAR, EPIC and ACT-R/PM that are used in the domain of HCI with a strong emphasis on ACT-R/PM which has been used for this research.

The architectures that are discussed in the following sections have much in common. They are all based on a production system and they all provide a goal-orientated approach when running a cognitive model. They are also based on cyclic execution with an execution time fixed per default to 50 milliseconds. Finally these cognitive architectures provide means to interact with an environment to some extent with some having more limitations than others. This discussion explains and provides the mean to determine by comparison which framework was the more appropriate to use for this project.

### 3.3.1 SOAR

The SOAR cognitive architecture developed by Laird, Rosenbloom, and Newell (Laird 2012) on the back of the research carried out on the General Problem Solver (GPS) (G. M. Olson & J. S. Olson 2003) around 1960. In short, the main research focus of GPS was on problem solving and the ability of humans to search through a problem space to solve problems using *means-ends* analysis; a common-sense approach where the available *means* are the operators that are selected to reduce the difference between the current situation i.e. the current mental *state*, and the goal to achieve i.e. the *ends* of the problem. The operators are the means by which the system can move through the problem, and the problem space consists of the operators and the set of possible states that can be visited to solve the problem (Laird & Rosenbloom 1992). GPS led to the research that brought the Model Human Processor (MHP) and the publication of *The Psychology of Human-Computer Interactions* by Card, Moran and Newell which led to the concept of GOMS, as the P9 Problem Space principles (Card et al. 1983).

SOAR inherited this concept. Based on a production system, it has two types of memory: a production memory and a working memory. The productions contain association between conditions and actions and represent both declarative and procedural knowledge. They are similar to productions rules that are presents in other production systems; namely they have conditions and actions. These rules are stored in the long-term memory.

Soar working memory includes the current problem solving state and operator (including sub-states that may arose during impasse process) and has also a short-term memory.

It contains elements called Working Memory Elements (WME). Several of these WME can be associated with the same factual knowledge and collectively fully describe this piece of knowledge. For instance we may have the fact about an object that is a button: “BTN1 is a button”, “BTN1 is a square”, “BTN1 is named OkButton”, “BTN1 is disabled”. All these WME’s are contributing to the knowledge SOAR has on BTN1 that will be internally refer to as BTN1, which is known as the *identifier* for BTN1. The group made of WME’s that share the same identifier i.e. BTN1 in this example is called an *object* with each WME describing a different *attribute* of this object. Therefore all WME with the same identifier are part of the same object. Objects in working memory are linked to other objects by setting a WME attribute set to an object identifier. In addition, all objects must be linked to a state through other objects, either directly (when doing problem solving) or indirectly (when accessing external sensors for example).

Similar to ACT-R and EPIC, Soar is based on a production cycle but it called the *decision* cycle. During this cycle Soar attempts to determine what changes in the system state can be made to progress in the problem solving operation. It comprises two phases: the *elaboration* and the *decision* phase. During the former one, all instantiated productions i.e. productions that matches the elements in working memory, fires in parallel and all the productions that are not firing are retracted. During the *execution* process, these rules can change the working memory or create preferences. This is the *preference* phase that suggests which operator to apply. Following this comes the working memory phase where all non-operator (tagged i-supported) preferences are considered and then evaluated leading to elements either added or deleted from working memory. The *elaboration* phase repeats until no more productions are eligible to fire or to be retracted hence making sure that all relevant available knowledge is taken into account during the decision. During this phase, some productions have fired, and a decision is taken as to which operators to select but no actions take place. This is during the following phase that operators are selected. All operator preferences (tagged o-supported) are considered and then evaluated. Concluding this phase the selection of a new operator or a new state takes place.

There are times that after a decision phase, no appropriate operator is selected. In Soar it is called an *impasse*, which is dealt with by creating a new state added to the working memory in which the goal of the problem solving becomes solving this impasse. During solving impasses, Soar is given the opportunity to learn. This automatic process is called chunking, in which a new production rules i.e. a chunk is created where the conditions of the chunk are the elements of the state that (arising from a chain of production firings) allowed the impasse to be resolved whilst the action of the production is the WME or

preference that resolved the impasse. During the *chunking* process, Soar creates variables based on the conditions and the related action so that when a new similar situation arises, this production may match and fire thus avoiding future impasses from arising (Laird and Rosenbloom 1992; Laird and Congdon 1998; Gray and Altmann 1999; Schultheis et al. 2006; Byrne 2008).

In terms of HCI, the inclusion (or re-inclusion when Soar was ported in C) of a graphical interface using Tcl/Tk called Tcl/Tk Soar Interface (TSI) allows model designers to develop explicit GUIs as external environments that can deal with human interaction (Ritter et al. 1998).

### 3.3.2 Executive – Process Interactive Control (EPIC)

The EPIC cognitive architecture developed by Kieras & Meyer (Kieras and Meyer 1997; Kieras et al. 1997) in an effort to reduce the design time for computer interface, and to evaluate analytically these interfaces without often having to spend time and financial resources for empirical user testing, and provides a platform to use engineering models for human computer interaction. It is a cognitive architecture for constructing models of human performance, which represents in detail the perceptual, motor and cognitive constraints on the human ability to perform tasks using computer interfaces. In short, EPIC goes beyond providing the type of engineering predictions of human performance as seen in a real world task domain.

EPIC in addition to facilitating the modelling of human cognition, perceptual and motor functions, provides a framework to simulate the interaction of humans with external systems and provides a task environment that can assign physical locations and properties to interface objects and generate events in response to simulated human behaviour.

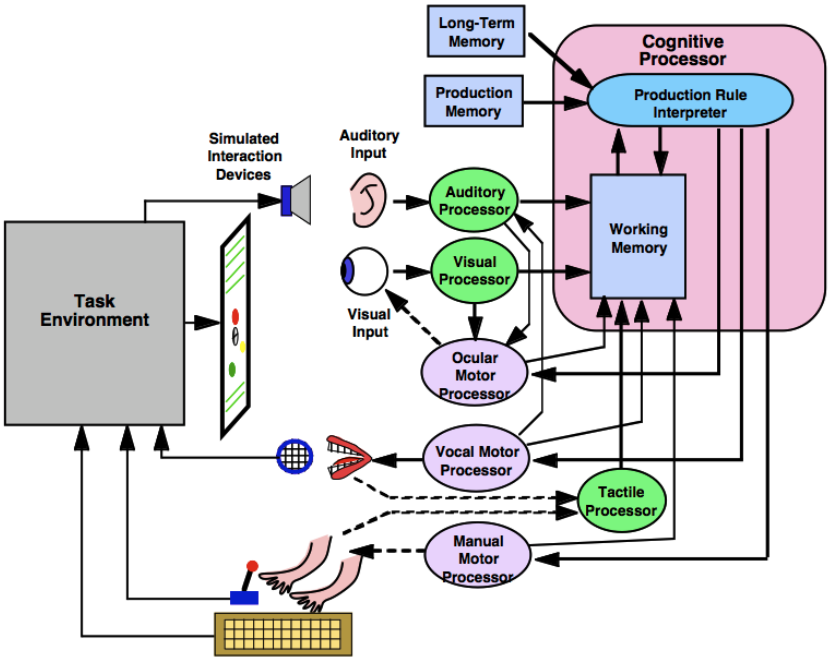



Figure 19: EPIC architecture (Kieras 2009)[29]

EPIC's memory system is subdivided into several working memories. There are memories for the perceptual processors i.e. visual, auditory and tactile memories that contain the current information generated by these respective processors. There is also a partition dedicated to the motor processor called the motor working memory, which holds information about current statuses and state of the processor i.e. whether a hand movement is being undertaken or has completed. All this information held in memory is refreshed on every cyclic turn. EPIC has also two other types of memories that do not contain information pertaining to sensory or motor processors. One is the *control store* that contains information representing the current goals and sub-goals to accomplish the designated goals. As EPIC used the Parsimonious Production System (PPS), this is an important addition as PPS stipulates that the control information should be simply be another kind of memory item which can be changed by production rule actions (similar to ACT-R), hence is a critical functionality for modelling multi-tasking performance as the production rules can control sub-processing simply by manipulating the *control store* content. The second *amodal* memory is used by EPIC to store miscellaneous task information. Unlike cognitive architectures such as ACT-R, there are no assumptions regarding memory decaying in the EPIC framework.

[29] Taken from "Why EPIC was Wrong about Motor Feature Programming" (Kieras 2009)

As already mentioned, EPIC largely inherits from PPS concepts and thus is based on PPS production rules which have the format (<rule name>IF<conditions>THEN<action>). The rules are not limited to initiating motor actions only but can add or remove items from the working memory. Those production rules contain clauses i.e. conditions that must be met. EPIC operates in terms of cycles. This implies that at the beginning of each of these cycles, the production rules are updated, and then their conditions are matched against the whole set of the current rule clauses. If these conditions contained in a rule matches the conditions the rule fire and its action part is executed which typically involves changing the content of the procedural rule memory, this influencing the flow of the rule firing process during the next cycle. Since EPIC follows closely PPS, every rules that matches fire during the next cycle in a parallel manner; thus simulating a parallel cognitive architecture; unlike known traditional production systems i.e. SOAR or ACT-R that require that only one production rule is fired at a time i.e. sequentially; controlled by some kind of conflict-resolution mechanism that compute which rule is best to fire; a process often constituting a bottleneck in terms of action executions.

The research presented by this thesis focuses only on the visual and motor with their associated cognitive functions, thus the discussion of the processors will concentrate only on these areas.

EPIC was one of the first frameworks to include relatively complex visual processing which takes into account the fact that human vision is not homogeneous and its acuity is highest towards the centre of the Fovea and decreases towards the periphery of the retina. The visual processor simulates the retina of an eye. It determines the kind of sensory stimuli information that is available about the objects contained in the *task environment* based on angular distances on the retina between an object and the centre of the fovea. This computation in EPIC is called the eccentricity. The eye model contains three approximated zones i.e. Fovea – 1 degree, Para fovea – 7.5 degrees and the retina periphery set to 60 degrees that are simulated by specific processors for each of those areas.

In the EPIC vision system the visual perceptual processor maintains a representation of object visibility and metrics in the visual working memory that are slaved to the output of the eye. This implies that it is kept automatically updated by changes in the scene as objects appear, disappear, and change their colour or when an eye movement or object movement occurs in that scene. In response to changes, events are issued and lead to

outputs with different timing tags by the visual processor depending, which zones the detection happens to be in with regard to the eye.

The EPIC framework contains many motor processors producing a variety of simulated movements for different effector organs; each having appropriate timing characteristics for their movement completion. Different processors exist for the hand, vocal and eye organs all of which can be activated simultaneously. The motor processors get inputs from the cognitive processor as a result of a procedural rule firing. The motor action is executed and the cognitive processor is kept updated about their state as motor clauses in the production rule memory. All motor processors have the same structural arrangement and share most of their functionality, i.e. in terms of coding they all inherits from a base processor class that contains most of the functionality. The movements for those processors are described by a set of features; the *style* feature being the most important that represents the kind of movement it relates to i.e. its name. The features for a style are represented in a hierarchical fashion that controls which specific effector of the motor action that must be generated. Thus when a feature changes from a previous one, the implications are that all the sub-features for that movement style change too i.e. a change in feature of the Punch action filters down changing the feature of the effectors such as the Hand and Finger also features.

The motor processors are based on a three phases cycle. The first phase known as the *preparation* phase generates a list of features needed by the movement and a time stamp of 50ms is applied for each feature that is generated. Then the movement *initiation* phase follows and sets the movement up which EPIC also assumes a time stamp of 50ms. Finally the *execution* phase is launched which consists of making the actual mechanical motion for the movement but the time stamp for this process depends on the movement itself. The manual processor, which controls the hands and fingers, is able to simulate four main styles of movement. The Punch style simulates high-speed response where a button is located directly underneath the finger. The Ply or Point styles simulate the moving of an object using a hand i.e. a cursor moving towards a target object, by manipulating a control such as a mouse or a joystick; the Point style being more specific to mouse and computer interface usage. Both these styles shared very similar functionality whereby the Point style inherits most of the Ply functionality. Both types of movements have similar *preparation* phase to prepare features such as *Direction* and *Extent*. The *Direction* feature is only computed if the difference between the new and old direction is larger than PI/4. Similarly the *Extent* feature is re-calculated if the difference between the new and old extent is over two degrees of visual angle. Once the preparation phase is completed, the initiation phase

proceeds with the calculation of target distance and size to compute the movement time using the modified version of Welford's modification of Fitts' Law, amongst other processes. Finally, the movement is scheduled for execution during the execution phase. The other style that EPIC provides is the Keystroke movement based on GOMS-KLM which simulates hitting a key on a keyboard. During the feature preparation phase, no time is stamped if the previous movement was a keystroke to the same key. This is adjusted to 1.5 features to prepare if a new key is hit and 3 features to prepare if the previous movement was not this style. The Keystroke style does not require a preparation phase before the action is scheduled for execution. The Welford formulation of the Fitts' law is:

$$time = \max\left(t_m, k\ log\left[\frac{d}{w} + 0.5\right]\right) \tag{3.1}$$

where $t_m$ is the minimal aimed movement time (default set to 100 milliseconds), $k$ is a constant specific for a style of movement i.e. for example for a peck action it is 75 milliseconds and for ply action it is 100 milliseconds. $d$ is the distance to travel to the target and $w$ is the width of that target. This implies that no movement can be performed in less than $t_m$ i.e. 100 milliseconds. The other alternative being that the duration of the movement is a function of the ratio of the distance to travel to the width of the target. The width is not simply the width of the actual target but is computed by taking the width of the cord through the target region taken on the line drawn from the starting point of the movement through the centre of the actual target. See Figure 20.

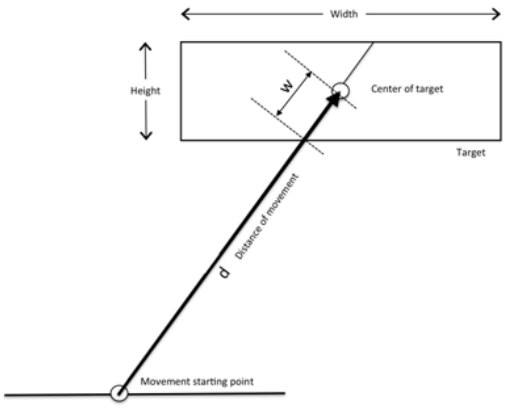



Figure 20: Welford formulation of Fitts's Law

The EPIC ocular system can produce both voluntary and involuntary eye movements that are initiated by the cognitive processor. These movements are composed of a number of saccades (or small movements) moving towards a target object. During the preparation

phase, similarly to the Ply style movement, both the direction and extent features are prepared using the current position of the eye to the target object. The time stamp applied to each saccade is four milliseconds per degree of visual angle. The oculomotor system can also initiate involuntary eye movements either through saccades or small smooth adjustments in response to changes in the scene. Therefore if an object appears in the scene, the processor will trigger an involuntary saccade helping the eye to adjust to the location of the object. By using a combination of both voluntary and involuntary eye movements EPIC can track a moving object.

Finally, in the EPIC architecture, the motor phases can be overlapped to bring efficiency to the motor system compared to phases executed purely sequentially by overlapping the initiation and execution phase of the movement with the preparation phase for the next scheduled movement (Kieras and Meyer 1994; Kieras and Meyer 1997; Kieras et al. 1997; Bryson 2001; Meyer et al. 2001; Kieras and Meyer 2004; Schultheis et al. 2006; Byrne 2008; Kieras 2005; Kieras 2011).

### 3.3.3 Adaptive Control of Thought—Rational (ACT-R)

The research presented in this thesis is based on using ACT-R/PM, which combines the cognitive module and both the perceptual and motor modules into one framework; the engine behind the visual module is based on Salvucci's research called *Eye Movements and Movements of Attention* (EMMA) (Salvucci 2001). Anderson stated in his book "Rules of the Mind" four basic concepts, which underline ACT-R. Some of these principles express the architecture as a production system that should be based on neural-like processes to fit in with the human brain function and finally to be able to deliver optimal behaviour given the statistical nature of the environment i.e. based on probabilistic principles through the use of Bayesian techniques as seen in ACT-R (Anderson 1993). Like other architectures, ACT-R is a goal orientated production system based on PPS but unlike these frameworks it combines both a symbolic and a sub-symbolic component with the latter tuning itself to the structure of the environment. Both components are using Bayesian learning mechanisms for the declarative and procedural memory. ACT-R goes beyond other architectures as it has embedded a sub-symbolic level composed of a set of variable quantities that are updated continually by ACT-R learning mechanisms to bring the qualitative side of human cognition thus tuning the performance of the human cognition in a neural-like activation manner in terms of activations levels. The activation levels determine the speed and ultimately the success of memory access for both declarative and during conflict resolution for procedural knowledge. The availability of a memory chunk for retrieval is established by these activation levels that are learned by ACT-R cognitive

framework according to statistics returned by the Bayesian networks derived from the history of use of the information contained in memory chunks. The chunks with higher activation are selected by ACT-R for retrieval (Lebiere et al. 2002). Like other architecture ACT-R has a cycle timing of 50ms (Byrne and Kirlik 2005).

ACT-R version 6.0 is the latest version of the ACT architecture, the latter first introduced in 1976 based on a production system as described by Newell. The evolution of this framework shifted when the rational analysis was added in ACT-R 2.0. Some of the major changes between the two versions brought partial matching. When one looks at a person's face for instance; which changes over time, one still recognises this person many years later even though those changes can be profound. This concentrates on calculating a degree of mismatch amount, which is then taken off the activation level, thus restricting chunks that can be partially matched to the ones that have the same type which themselves compete for a match to a production rule condition, thus impacting on the retrieval probability of chunks. Another change is the way the production rules are selected. In ACT-R 2.0, all productions were contenders for selection whereas in version 4.0 only the ones whose top goal matches are selected for the conflict set; thus reducing excessive pattern matching. Attempts to retrieve chunks are made to match the rest of production conditions when the conflict set is created. In version 2.0, this process did happen during the construction of the conflict set. Another major change was the change of the learning mechanism from analogy where past examples are used to provide likeness between past and current problem solving goals. This mechanism produced many difficulties such as an over-creation of production rules. This complication was resolved in ACT-R 4.0 by introducing the creation of declarative knowledge called *dependency* goals in a process called production compilation. During this process new production rules are derived from declarative chunks immediately created upon the popping the dependency goal when the goal is achieved and added to the production system. In addition, this generalisation process created variable objects of any memory chunks that occur in two or more slots anywhere in the condition or in the action part of the created procedure rule. The next iterations of ACT-R saw an extension that incorporated perceptual and motor processes resulting in the version named ACT-R/PM (Anderson 1993). ACT-R/PM combines ACT-R, a visual interface specially adapted for ACT-R created by Mike Matessa (Anderson et al. 1997) and the motor section of the EPIC architecture (Kieras 2011).

Both ACT-R 5.0 and 6.0 brought to the architecture a better integration of cognition, perception and finally motor processes. In addition, the goal stack that existed with ACT-R 4.0 was removed leaving the task of goal setting to the action part of production rules. The

research presented and discussed in this thesis is HCI orientated and based on ACT-R/PM 6.0 architecture. The conclusion of this section will therefore focus on the functionality that pertains to the functionality needed in the TOISE system.

Before describing the visual and motor modules in some details, some attention is given to buffers in ACT-R that are somehow windows to each of those modules.

#### 3.3.3.1 Buffers in ACT-R

In terms of HCI, the interest of modelling often comes to areas dealing with memory handlings, visual, auditory or motor functions. ACT-R contains a set of buffers for each of those modules, which as already mentioned, enables asynchronous operations to take place. Any changes to a module buffer will immediately been followed by a module response resulting in the generation of event(s) to perform some type of actions and if appropriate the module might place a chunk in its buffer to signal the outcome of these actions (Bothell 2008). Buffers have two main functions in ACT-R, which are making requests to a module from a production rule and holding a chunk. One of the main tasks that are often carried out in ACT-R modelling is memory retrieval. Like motor commands, memory retrieval is initiated on the right hand-side of a production rule. The pattern matching that attempts to match the conditions specified on the left-hand side of a production rules to the current content of the specified buffer as we have discussed previously is carried out through those buffers and this point is illustrated with the production rule listed below:

```
(P ATTEND-OBJECT-A-LOCATION
      =GOAL>
            ISA             GOAL
   STATE            ATTEND-LOCATION-A-ACTION
 =VISUAL-LOCATION>
            ISA             OBJECT-A-LOCATION
 ?visual>
            STATE             FREE
==>
 =GOAL>
            STATE             LOOKFOR-VISUALOBJECT-ACTION
      +VISUAL>
            ISA             MOVE-ATTENTION
            SCREEN-POS        =VISUAL-LOCATION
 +VISUAL-LOCATION>
            ISA             OBJECT-A-LOCATION
)
```

There are two sides to a production rule. The left-hand side i.e. LHS which is the content that is above the "==>" and the right hand-side i.e. RHS is below that line. A rule contains a name, which in Lisp is a symbol. In the example above, the name is "ATTEND-OBJECT-A-LOCATION". This name must be unique. The left-hand side of this production rule is therefore:

```
=GOAL>
            ISA             GOAL
  STATE             ATTEND-LOCATION-A-ACTION
=VISUAL-LOCATION>
            ISA             OBJECT-A-LOCATION
?visual>
            STATE           FREE
```

This part can be interpreted in plain English as follow. If the goal buffer contains a memory element of type "GOAL" and its state slot is "ATTEND-LOCATION-A-ACTION" i.e. if the goal is to attend location "A" and the visual buffer contains a memory element of type "OBJECT-A-LOCATION" i.e. if the object to attend is the object present in location "A" and the state of the visual module is free then this production rule can participate in the conflict resolution process. In ACT-R, all production rules whose LHS that are in the procedural memory that match against these conditions will be put into the conflict set with the one with the highest utility set to fire.

In a production rule the RHS initiates modifications of the buffers. These changes can lead to a request of changing the information contained in the current buffer or a request to execute a command that is associated with it. In the right-hand side such a request is marked by a "+" sign in front of the buffer name this change pertains to. This is shown in the production rule below:

```
=GOAL>
            STATE           LOOKFOR-VISUALOBJECT-ACTION
    +VISUAL>
            ISA             MOVE-ATTENTION
            SCREEN-POS      =VISUAL-LOCATION
+VISUAL-LOCATION>
            ISA             OBJECT-A-LOCATION
```

In plain English this RHS can be interpreted as follow. In short, set the goal to look for an object and change the visual buffer to issue the command for the visual module to move its attention to the visual location where object "A" is located. This is achieved by setting the state of the goal to "LOOKFOR-VISUALOBJECT-ACTION", setting the visual buffer type slot to "MOVE-ATTENTION" and its screen position to the visual location "OBJECT-A-LOCATION".

#### 3.3.3.2 Vision functions in ACT-R

The vision module provides the functionality pertaining to vision on the device. It sees the features that can be attended to by a model, which have been added and then provided by the device. In ACT-R/PM the set of features that are presented by the device is named the *visicon*. These features provide information such as the type, location, colour, kind and so on. The Vision system is composed of two major parts named by Dan Bothell as the

*where* and *what* systems. The *where* system (or feature search) is the side of the vision module that takes requests passed through the visual-location buffer. A typical request is similar to the one shown below which instructs the system to find a feature whose type is named as "OBJECT-A-LOCATION" making sure it is attended. Responding to this request, the system updates its visual-location buffer by creating and placing a declarative memory chunk into that buffer that contains the location of the object in the device if it found. The attended slot is important in modelling vision as they are markers that have limitations in both their duration and numbers i.e. there are so many items that can be attended at any one time and their duration is limited either by a voluntary shift of attention to another object or naturally i.e. one can only attend an object for some time only.

```
+VISUAL-LOCATION>
    ISA         OBJECT-A-LOCATION
    :attended   t
```

The *what* system's (or movement of visual attention) main interface to a model is the visual buffer. Its main function is to attend to locations in the device that have been previously found with the *where* system. To attend a target, the model needs to provide a chunk that describes the visual location of where that target is located. If an object exists in this location, the system will respond by placing a chunk in its *=visual>* buffer representing the object. The system has an inbuilt tolerance set by default to 0.5 degrees of a visual angle, which ensures that if an object has moved a little since the time it was first located, it would still be attended to. For instance, a statement shown in the code listing below tells the vision module to shift its attention to the object that is located at "=VISUAL-LOCATION".

```
+VISUAL>
   ISA              MOVE-ATTENTION
   SCREEN-POS       =VISUAL-LOCATION
```

In the LHS of a production rule, the chunk that is placed in the visual buffer can be tested using the matching clause as shown below. For instance in this example, we are testing for a visual object that is of type "OBJECT-A" and named "object1".

```
=VISUAL>
  ISA              OBJECT-A
  VALUE              "object1"
```

The visual-location buffer can be tested for errors or attention states. This is done using slots such as Attended or State error. The former can be *t* if attended, *nil* if it is not or new if it's just been added to the model *visicon* and not yet attended. For errors, the state can be true i.e. *t* if a request has failed or *nil* otherwise. In the same manner the visual buffer can be tested for errors and states. For instance the state of the visual module can be *busy* if a visual request was initiated but not yet completed and *nil* in other situations. A

state can be *free* if a visual request has been initiated and just completed for instance, *nil* otherwise. Similar to the visual-location buffer, an error state exists. Finally a change in the device i.e. an object just appears in the scene, the Scene-change state reflects this change by setting the slot value to true i.e. *t* or *nil* otherwise.

#### 3.3.3.3 Motor functions in ACT-R

The motor module is based on EPIC Manual Motor Processor from Kieras and Meyer (Kieras and Meyer 1997) and can be perceived as the model's hands. It provides the functionality to interact with a keyboard and a mouse in the same manner as a real user and can be extended to interact with other devices through the creation of proprietary modules. Like other modules it has its own buffer from which it determines which action to perform. Unlike the visual buffer, the buffer associated with the motor module does not update this buffer in response to some required action. The keyboard that is associated with the motor module is a standard QWERTY keyboard that is often seen as part of standard computer station equipment. The mouse is controlled using the right hand is a fairly old model with one button only. However, this is sufficient for most modelling work dealing with HCI. In order to make a movement using the keyboard or the mouse, the RHS of a production must make a request of a specific type of movement called a style. The motor module inherited the movement styles provided by EPIC Motor Processor. The primitive movements provided by the motor module are listed in the table below.

| Action | Definition |
|---|---|
| **Punch** | This action is similar to the action when one presses a keyboard key when the finger is directly above the key. It is a simple down-stroke movement followed by an up-stroke one. |
| **Peck** | This action is in actual fact two movements combined together. The first movement is a specific movement towards a location that is followed by a keystroke. |
| **Peck-recoil** | This action a combined action containing a Peck action that is followed by a movement that returns the finger to its starting position. |
| **Ply** | This action is similar to the action one does when moving a device such a mouse to a specific location. |
| **Point-hand** | This action moves the hand to a specific location. |

The motor module provides more complex movements in addition to those primitives. For instance, there is the mouse-cursor action, which is a combination of the "ply", and a press-key action for the right hand; the latter being either a "punch" or "peck-recoil" action.

Similar to EPIC, there are three phases or states when performing motor movements i.e. preparation, initiation and execution.

The preparation phase builds a list of features that are used to guide the movement. The maximum number of features is five. The timing associated with this preparation phase depends on the amount of features that needs to be prepared and increases with the complexity of this preparation. Each feature takes 50 milliseconds to prepare. The feature set of calculations takes into account the complexity of the movement on one hand and refers to the previous movement on the other. More complexity means more needed features to be computed and more similarity with the previous movement results in less features to take into account during the calculations. The motor module keeps a record of the last set of features that was prepared for the previous movement in its buffer.

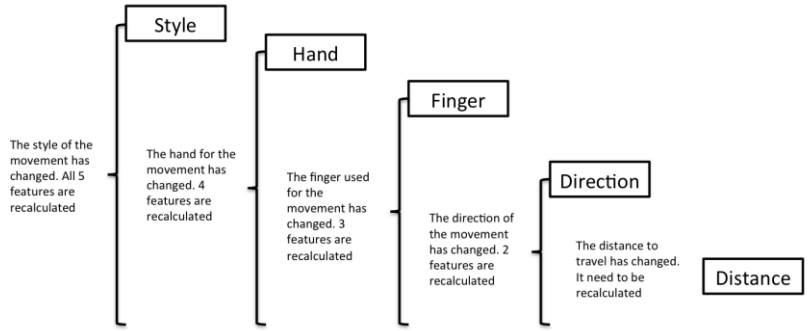



Figure 21: Feature calculation during preparation of motor movement

There is a hierarchical structure about features as shown in Figure 21. All movements for instance have a style and all have a hand feature. If the style of a movement changes, all the features for this movement are therefore re-calculated i.e. a different movement altogether. If the style of two sequential movements is similar but the hand changes, the features for the style do not need to be recalculated but the feature for the hand and the ones below do require recalculation. Thus, four features needs to be computed. The remaining 3 features are often specific to the type of movement requested. For instance when using the keyboard and punching a key, only the hand and finger are needed to perform this movement. During this phase the preparation state becomes busy. It is worth noting that like in EPIC, the direction feature is only computed if the difference between the new and old direction is larger than PI/4. Similarly the extent feature is only re-calculated if the difference between the new and old extent is over 2 degrees of visual angle (Byrne 2001).

| Action | Features needed |
|---|---|
| **Punch** | This style of action is a simple action that needs only 2 features i.e. a hand and a finger. However, in terms of execution time, punching a key of a keyboard also must take into consideration the mechanical side i.e. it takes time for the switch of a button / key to make contact. This is termed as the *closure time*, which is estimated at 10 milliseconds, which is added to the final time. |
| **Peck** | This style of action is a directed movement of a finger to a target location that is followed by a keystroke that needs 4 features i.e. a hand, a finger, a direction and a distance. In terms of execution time, this action involves the Welford version of the Fitts' Law using a *k* constant of 75 milliseconds. See section on EPIC and Figure 20 for more details on Fitts' Law |
| **Peck-recoil** | This style of action is identical to the Peck style with the difference that the finger returns to its original location. Thus 4 features are needed and in terms of execution time, like the Peck style, this action uses the Fitts' Law with added consideration that the returns back to its original starting point. |
| **Ply** | This style of action refers to a hand movement that holds the mouse to a new location that needs 3 features i.e. a hand, a direction and a distance. In terms of execution time, this action involves the Welford version of the Fitts' Law but using a *k* constant of 100 milliseconds. |

Once the proposed movement has been prepared, the motor module makes the specified movement. When it performs the movement, it undergoes an initiation phase that sets the movement up with an assumed 50 milliseconds time stamp. During this phase both the processor and execution states becoming busy and when it finishes, the processor state is set to free again but the execution state remains until the movement is fully completed. Following this phase the actual execution starts. The overall amount of time that a movement takes to execute this action is mainly based on the style of the movement and the distance to travel to achieve the movement. Complex movements such as moving a mouse takes longer to achieve as their execution time is based on Fitts' Law (See section on EPIC and Figure 20 for more details on Fitts' Law) compared to punching a key on a keyboard or the mouse button for instance. As already mentioned, the motor module is based on EPIC's Manual Motor Processor. This implies that it can execute an action whilst preparing the next one but it can only prepare them sequentially. Hence, if an extra movement is scheduled at the same time and therefore needs to be prepared, the motor module is said to be "jammed". If it is already busy preparing a movement and an error is generated this results in a notification in the ACT-R/PM console. To avoid this jamming situation, the model production rule that initiated the movement should have a constraint in its LHS testing for the state of the motor module as shown in the code below. In simple terms, the constraints stipulate that if any procedural rules have a goal to press a mouse and if the motor module is not busy, then add this rule to the conflict set.

```
...
=GOAL>
   ISA        GOAL
   STATE      PRESS-MOUSE
   ?manual>
     STATE    FREE
 ==>
```

...

### 3.3.3.4 Memory retrieval in ACT-R

Another important aspect of the cognitive architecture ACT-R is the way declarative memory elements are retrieved. Not only ACT-R has a symbolic system of production rules but it also implements a sub-symbolic activation based memory system, which influences the retrieval of memory chunks.

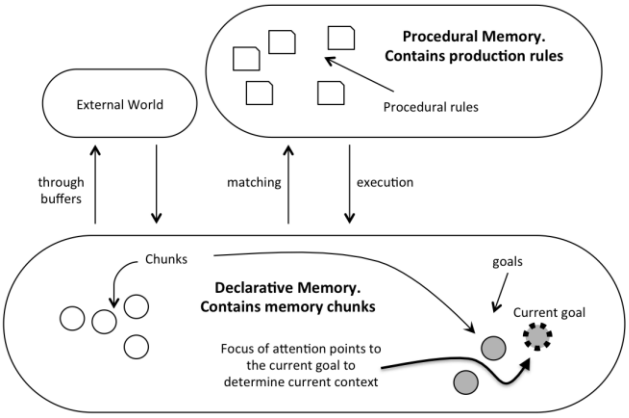



Figure 22: ACT-R Memory organisation

Unlike SOAR, ACT-R does not have a separate working but instead it has a short-term memory (declarative memory) and a pointer called Focus of attention that points to goal chunks stored in that memory to keep track of the current context (See Figure 22).

New declarative memory chunks are added to the declarative memory either when goals are created through the action of production rules (creation of declarative goal chunks that are stored in the goal buffer, effectively replacing the old existing goal with a new one) or being possibly encoded as a result of changes in the external world. Each chunk has an activation value which reflects its previous use and association with the other chunks contained in memory. This activation value reflects the probability that the chunk will be needed in the current context. Thus if it is the case, each time a chunk is retrieved from memory, its activation value is increased. Chunk activation is also subject to decaying so the longer ago a memory chunk is activated, the lower its contribution to the activation will be. In addition there is an activation threshold beyond which chunks cannot be retrieved.

The most appropriate method to explain this activation concept is by means of an example. Suppose that the task at hand is to solve an addition problem such as "*7 + 3 = ?*" as shown below.

```
=goal>
   ISA      addition
   arg1     7
   arg2     3
   sum      nil
```

All chunks that have an association with 7 and 3 will have a better probability of being suitable to solve this problem. Let us suppose that previously a subtraction problem was completed such as 10-3=7 as shown below:

```
=goal>
   ISA      subtraction
   arg1     10
   arg2     3
   sum      7
```

This chunk would have been pushed into declarative memory and in the current context this chunk in memory will get an increase of its activation through 3 and 7.

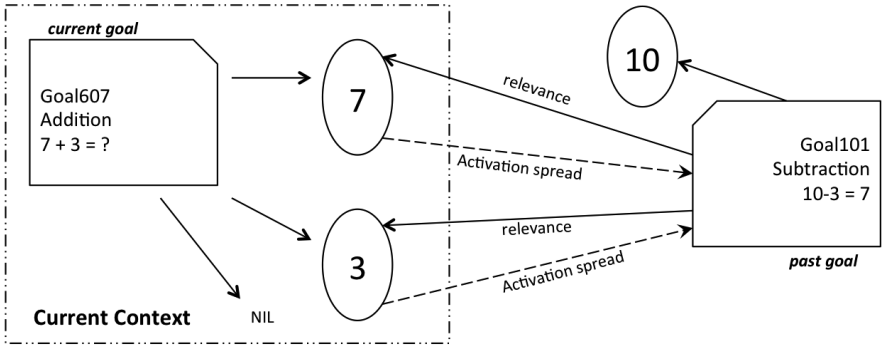



Figure 23: Activation process in ACT-R

Looking at this example and Figure 23 one can determine the relevance of the two chunks towards each other. It is however the sub-symbolic activation and spreading system that creates this relevance in ACT-R (Taatgen 1996).

The formulation for the chunk activation is given by this formula:

$$A_i = B_i + \sum_j W_j S_{ji} + \varepsilon \qquad (3.2)$$

$A_i$ is the activation value of chunk $i$. The computed activation value has two main parts; a fixed base-level activation $B_i$, which is specific to chunk $i$ that reflects it general usefulness in the past and can relatively vary given some random noise $\varepsilon$ to reflect stochasticity in the nervous system. The second part is the variable part is determined by

the current context reflecting its relevance to it. What the summation does is adding up the influences for each chunk in the current context. $W_j$ determines if a chunk has influence over chunks in current context. ACT-R assumes that there is a fixed capacity for source activation and that each chunk in the context has an equal amount with the overall amount set to the value of 1 as default. Thus, if there are $n$ chunks in the current context, each influential chunk gets $1/n$ source activation. If a chunk however does not influence chunks present in the current context, then $W_j$ is set to zero. $S_{ji}$ represents the strength of association between chunks. It is an estimation of the odds of *j* being a source present if the chunk $i$ is retrieved (Anderson and Lebiere 1998).

Chunk activation levels have substantial consequences with regards to memory retrieval. First in situations when more than one chunk matches a procedural rule constraints (or partially match) the one with the highest activation is retrieved. In case of partial matching, situations can arise when high-activated chunks are partially matching procedural rule constraints – in this case an activation penalty is imposed, but this can still lead to a partially matching chunk to be selected in favour of a fully matching one. The final point to highlight is that chunk activations are tied to memory retrieval. The time it takes to retrieve a chunk from memory by a procedural rule relates mainly to two factors i.e. the match score of the chunk and the production rule strength. The match score is the activation value minus the degree of mismatch. Consequently, if there is not partial matching then the match score is the activation itself. In this case the formula based on an exponential function is:

$$Time_{ip} = \mathrm{F}e^{-\mathrm{f}(A_i+S_p)} \tag{3.3}$$

This represents the latency needed to retrieve chunk $i$ by the production rule $p$ and Both F and f are scaling constants. $S_p$ is the strength of the production rule that is attempting to retrieve the chunk and $A_i$ is the activation for chunk $i$. As the sum of the rule strength and activation decrease, and thus their relevance decreases, the time to retrieve a memory element increases exponentially (Byrne 2008).

#### 3.3.3.5 Conflict Set in ACT-R

In early chapters we discussed procedural rules that construct a model. Bearing in mind that ACT-R is goal directed the decision to select which of the rules to execute is referred to as the conflict resolution and all the procedural rules that are matching on the basis on goal matching are entered into the conflict set. This *conflict set* may contain one

or more productions with only one of which that will ultimately be selected for execution. When a production rule is selected from the conflict set, and then the rest of memory retrieval is completed (if required). ACT-R maintains a number of parameters and value attributes for each production rules to calculate the likelihood that a production is selected.

Productions in the conflict set are ordered in terms of their expected computed gain. The equation for this computation is:

$$E = PG - C + \varepsilon \tag{3.4}$$

Where $E$ is the expected gain or utility of the production rule, $P$ is the probability of success that the goal will be achieved if the production is selected, $G$ is the value of the goal and $C$ is the expected cost of achieving the goal if the production rule is selected. $\varepsilon$ is some stochastic noise to fit the decision in line with human behaviour. Both $G$ and $C$ are measured in terms of time. Thus $C$ is an estimation of the time it will take to achieve the goal and $G$ symbolises the worth of the goal or how much time ACT-R thinks it can spend on that goal. $PG$ becomes the expected gain if the production rule is selected. $E$ is therefore the expected utility of production as a trade-off between cost and worth of the goal. This implies (because of $P$) that the procedural rule becomes more selectable as the rule becomes more successful over time. This is, for instance, why the utility of production rules can be used to find the best strategy to adopt in experiments, for example, as Peebles and Bothell's response task that deals with sustaining attention (Peebles and Bothell 2004)

It is worth mentioning these computations only apply to selecting the best rule to fire in terms of goal matching. However if for some reasons the rest of the constraints for that rule fail to match once it is selected, the next ranked procedure in the conflict set is selected. This behaviour is repeated until a procedure is successful in terms of constraint matching. Another interesting point exists in which the firing of the procedural rules is completed serially whereas the conflict resolution process is mostly done in parallel.

Finally in the formula above, we used $P$ as a probability $G$ and $C$ as estimates. However $P$ is an aggregate probability of two probabilities namely $q$ and $r$; $q$ being the probability of the production to function successfully and $r$ is the probability of achieving the goal if the production rules function successfully.

$$P = qr \tag{3.5}$$

The resulting probability $P$ is the probability of success that the goal will be achieved if the production is selected. The higher this value then more likely the production rule will fire. Similarly, $C$ is an aggregate of two values namely $a$ and $b$. The higher this value is and the less likely the production rule will fire. The value $a$ represents the amount of current effort (as a latency) that the production rule will need to complete and $b$ represents the amount of time from the time the procedural rule complete until the goal is achieved.

$$C = a + b \tag{3.6}$$

The value $b$ takes into consideration the presence of sub-goals because their completion time is taken into account as well as any actions that must be completed (Anderson 1993; Taatgen 1996; Anderson and Lebiere 1998; Anderson and Schunn 2000; Johnson et al. 2002; Anderson et al. 2004; Stewart and West 2006; Bothell 2008; Byrne 2008).

$C$ and $P$ are important values in learning and problem solving in which cognitive modellers can forcibly alter those values during specific problem solving. Fu for instance uses this technique in a navigation task on computer-simulated map by removing success from the equation only focussing on cost (Fu 2001) as cost is the main factor as all paths in the map lead to the destination eventually (Fu 2003).

### 3.3.4 Summary

In this section we have discussed Soar, EPIC and ACT-R. Comparing those architectures and bearing in mind what TOISE has to fulfil, the factors provided by ACT-R led to this architecture to be used in this research. The perceptual motor functionality is well developed and includes better integration of eye movement using EMMA and interaction with environment i.e. it can interact with external environments and not just simulate them like EPIC (SOAR does not have this functionality integrated per default). It has both a symbolic and sub-symbolic components whereas the others architectures have only a symbolic one. In addition ACT-R has an integrated noise generation at the sub-symbolic level that reflects stochasticity found in human behaviour. Finally, there is a lot of support material for ACT-R such as documentation, tutorials and much published research work with source code available and visual environment that is very helpful to develop models. In summary, ACT-R was found better suited as a cognitive platform to use with TOISE.

## 3.4 Cognitive modelling used in HCI

This section discusses some past and present research and work that apply to the domain of HCI, in particular the instances that have common ground with the research this thesis presents in which cognitive models are used to simulate human users. In this section we will present and discuss research that was carried using ACT-R to simulate human cognitive, perceptual and motor functions in the domain of HCI often as substitutes for real users (Ritter and Young 2001). There is a trend towards applying cognitive modelling to real world problems, which include domains like HCI, often focussing on human interaction in real-time. In a research environment, different tools can be used to provide different required results, which can be combined together to produce final results. Attempts are now made to integrate all the necessary functionality into one development environment to extend the capability of the main system and integrating other tools that are well suited and validated for a specific domain (Lebiere et al. 2005). This effort often presents many challenges i.e. communication between the different modules for instance. The research presented in this section often has had to meet those challenges in the same magnitude as the research presented in this thesis.

### 3.4.1 ACT-Simple

Cognitive modelling is evolving and is used to predict user behaviour. Powerful techniques have emerged for exploring user interaction with complex systems. GOMS and its variant platforms such as KLM-GOMS or CPM-GOMS have been used to predict user behaviour and often sped up testing and evaluation of user interfaces but often lacks the precision to model detailed behaviour. The issue often encountered using these platforms is that they require expertise and are also time consuming.

ACT-Simple combines the simplicity of high-level frameworks with the power found in lower-level architectures by compiling basic cognitive and perceptual motor commands into ACT-R production rules. The available command set is limited but in terms of HCI is sufficient to provide a useful tool for predictions of human behaviour.

The process involves a sequence of KLM-GOMS like actions that are compiled into corresponding ACT-R procedure rules. An initial production rule is generated to initialise the current goal and from this point the rest of the generated rules are executed in the correct order until the end of the sequence of actions. ACT-Simple has a command set that includes: *move-hand*, *mouse-mouse*, *click-mouse*, *press-mouse*, *release-mouse*, *press-key*, *speak*, *look-at*, *listen* and *think*

ACT-Simple has been validated many times over a period of time and has been found to match empirical data with a percentile of errors around 10-12% but more importantly with an excellent correlation of 0.99. Thus the fluctuations of the data are possible in terms of human behaviour but the variability shows that data generated with ACT-Simple is very close to empirical data in terms of variations (Salvucci and Lee 2003). This compilation ensures that models that are developed in ACT-Simple inherits the features of ACT-R and in addition when run, they also inherit the parallelisation of perceptual and motor processes presented by this architecture, which thus becomes psychologically more plausible. It is worth mentioning that ACT-Simple has been used in many research and tools that will be discussed in the next sections.

### 3.4.2 CogTool

Engineering models of user behaviour have had a surge in popularity in HCI, but they are complex to build and this impacts on how widespread they are used. John & al 2004 in their research developed a system called CogTool (John et al. 2004) in which designers can generate predictive cognitive models simply by demonstrating the required tasks on a mock-up of intended interfaces.

The prediction of skilled performance in the HCI domain has been researched and validated by many research projects and yet its widespread use as a tool to help designers is somewhat discouraging. As John stipulated, even using KLM is cause for concern as it is often perceived as too difficult to model.

The first version of CogTool is somewhat the precursor of the now popular CogTool toolkit and uses Macromedia HTML IDE[30] as a base for constructing and instrumenting designs. The second version presents its own interface for design not relying on HTLM, Netscape and the other tools that were needed in the earlier version. CogTool is a combination of tools to predict latencies on user behaviour whilst demonstrating tasks on a mock-op design, which does not require programming cognitive modelling skills. There were two versions made over the years. The current version of CogTool is 2.0.

CogTool is based on Keystroke-Level Model (KLM) based on Human Information Processing (HIP) theory of human behaviour first proposed by Card & al (Card et al. 1980; Card et al. 1983) that is specific to user behaviour interacting with computer interfaces.

[30] Dreamweaver from Macromedia purchased by Adobe Systems in 2005.

It uses ACT-Simple to translate the instrumented actions into KLM rules which are themselves translated into more fine-grained production rules in ACT-R using Salvucci & al compiler (Salvucci and Lee 2003). Thus, movements that are analysed obey the Fitts' Law (see section 3.3.2) but in addition integrate in the timings the theories of preparation of actions before every task to be achieved as described in section 3.3.3.3. It incorporated significantly more detailed theoretical accounts of human cognition, perception and performance than the simpler KLM. It is a tool that produces quantitative predictions on how users will behave when the mock-up is finally implemented and gives an insight regarding cognitive load when users employ it (John and Salvucci 2005).

The earlier version included a Behaviour Recorder that observed a UI designer demonstrating tasks on a mock-up design run in Netscape using LiveConnect. LiveConnect is now discontinued since version 7.0 of Netscape but the concept at the time was noteworthy. The system presented widgets in a special palette that could be used to create a wide range of designs. Once the design was finished and demonstrated, the Behaviour Recorder (John and Prevas 2003) would translate the user actions into KML operators, which were then compiled into production rules.

Validation was performed and compared to latencies observed in previous publications so models generated by this system have been shown to be more accurate or at least accurate in mapping findings by other researchers. As john mentioned in her paper, the main drawback with this system is that four different systems must run in concert i.e. Dreamweaver, Netscape, the Behaviour Recorder and ACT-R in its Lisp console, which is also a problematic and somewhat cumbersome issue when it comes to developing user interfaces (John and Prevas 2003).

The next and current version of CogTool is inspired by projects such as SYLK and DENIM that both focus on sketching, storyboarding and designing applied to web content design for example (Landay 1996; Lin et al. 1999; Newman et al. 2003; Almasri 2005). Unlike its first version, the contemporary version has integrated a storyboard system, transitions between frames and contains widgets containing common components for interaction. The advantages being that the screenshots of existing products can be inserted as background images and those widgets can be located and overlaid on top of existing versions from the screenshots and thus CogTool can benchmark any existing product such as electronic devices, web sites etc. Additionally one can sketch ideas and use these ideas to profile cognitive load. As the early version, the interaction is done by demonstration to produce a computational cognitive load of a skilled user's performance.

Validation was performed and compared to latencies observed in previous publications and models generated by this system have shown to be more accurate or at least mapping exactly findings by other researchers (John 2008; John 2010).

An interesting development that used CogTool is the CogTool Explorer that predicts user exploration choices given a user interface and task, which attempts to predict what users might do when presented with a user interface to give prediction of user choice. The idea is to extend the existing CogTool, which predicts skilled performance on routine tasks and add predictions of user choices in exploratory tasks. It is based on SNIF-ACT (Teo et al. 2007). SNIF-ACT attempts to simulate users as they perform unfamiliar tasks such as searching while browsing Internet pages. It is based on information *scent*, which is based on spreading activation of chunks in declarative memory and ultimately affects the matching of production rules. The activation of content-dependant memory chunks that are matched by production rules can become the base for determining the utility of selecting these productions rules dynamically. The assumption behind this development is that activation spreads from the user's goal, behind the focus of attention, through memory associations to words and images that are visible on web pages (Pirolli and Fu 2003; Fu and Pirolli 2007; Teo et al. 2007; Teo and John 2011).

In terms of relevance to the work presented in this thesis, this early platform has common concepts. Firstly, tasks are demonstrated on a mock-up designed for which an automated cognitive model is generated and executed to gather user behaviour latencies. Secondly, the user interaction is in due course compiled as ACT-R production rules. However, the behaviour is compiled down from a high level using very few types of primitive motor or mental actions, to a lower one, which may restrict its use for more complex tasks.

### 3.4.3 Prediction of Driver behaviour and in-Vehicle interfaces

#### 3.4.3.1 Predicting the effects of In-Car Interface on Driver Behaviour

It is now common knowledge that attempting to manipulate an interface whilst driving has essential consequences which impact on driver performance whilst driving. Salvucci's research (Salvucci 2001) attempts to predict those effects. This research attempts to integrate into one system both a driver cognitive model that accounts for basic driving skills which has been well validated, and a user cognitive model that mimics user behaviour when using a device. It concentrates on dialling phone numbers whilst performing a driving exercise but could easily apply to devices, such as car radios and

other controllers that are often integrated as car instruments. The purpose of this research is to attempt to generate predictions focussing on interface interaction on driver performance and vice versa in terms of dialling numbers on a dialling pad.

Using the ability of ACT-R to interact with virtual interfaces, Salvucci constructed a simulated user that dials a number into four dialling interfaces hands-free cell phone that is mounted on the dashboard of the car with each one of them having benefits and drawbacks in his cognitive models that are integrated into the driver model already validated. The four phones are based on *fully manual* where the user keys in all the number of contact, *speed manual* where the user only keys in one key for that contact, *full voice controlled* where the user speaks the number to dial and *speed voice control* where the user only speaks the name of the contact to dial. The same contact is used for all experiments with the speed-dialling interface less demanding in terms of cognition as the full-speed dialling ones.

The main task is a usability experiment of four different interfaces while driving. A task analysis was achieved and a set of interactions with the cell phone was created with commands such as *Recall number*, *Press digit*, *Say digit* and many more. These commands provided the basic elements used to create the user model.

Because the two models are integrated into one, the main concern was to decide what to do in multi-tasking situations i.e. one can drive and still be able to press a power button on the cell phone. In this case, a lot of the available commands pass control over to the driver model after completion of the task.

The validated driver model controls movements in a sequential manner i.e. assess information and applies control. However, this model presents other aspect of driving including looking for blind spots etc. that affects the accuracy of the control. The research presented by Salvucci focuses on exploring the behaviour degradation in terms of control when an extra user interface is being operated in parallel.

The predictions being focussed on are dialling times and a measure of control accuracy i.e. with regards to the road lane and individuals were asked to perform 32 dialling experiments at the same time to compare them with results obtain from the cognitive model. Findings of the user testing, in comparison to model predictions, revealed that experiments validated known facts. Dialling times while driving are slightly longer mainly due to the control of the car and full dialling compared to speed dialling is slightly longer with a good correlation between model and user latencies. In terms of control of the car,

the findings found that human driver data presented greater deviation than the model, yet both correlated well but the model under-predicted the speed that dialling has on car control where it failed to capture its impact. In addition the results showed that manual interfaces are more taxing in terms of control of the car than voice controlled devices (Salvucci 2001).

This research and similar findings in this domain has shown the importance and contribution that cognitive modelling can make in research fields where high cognitive demand is required of users. One can also conclude that systems such as the one described can help designers to develop more usable interfaces.

#### 3.4.3.2 Distract-R: Rapid Prototyping and Evaluation of In-Vehicle Interfaces

Vehicle distraction is becoming a major issue in a society focussed on personal electronic devices and specifically car devices dominating our lives while travelling. Hence, researchers respond by engineering tools to build less distracting devices as publicity concerning the dangers associated with driver distraction, such as mobile phone use, are becoming more publicised. Distract-R is one of those tools to help evaluate new in-vehicle interfaces in terms of human interaction to steer away from the time-consuming and expensive modelling that is required to achieve this task. Driver simulators are often used to monitor user behaviour but studies using those tools are time-consuming and costly.

To identify the aspects of distractions, researchers have begun to use cognitive models in an effort to predict elements such as latencies of distraction (Salvucci et al. 2001; Salvucci 2002; Salvucci and Macuga 2002). As already mentioned, Usability Engineering is difficult as it demands highly trained professionals in the loop and thus becomes expensive. In an effort to address these issues, work has been carried out to use fast prototyping and modelling by demonstration (John et al. 2004).

Distract-R follows on this work and provides a prototyping and modelling tools that are less limited and focussing only on In-Vehicle interfaces.

Similar to the research presented by John & al work, Distract-R used a version ACT-Simple with a simplified driver model that has been validated in research such that presented in section 3.4.1.

As shown by John, Distract-R attempts to abstract cognitive modelling and is intended to aid designers and engineers and contains five distinct parts. The first tool is the rapid prototype component in which simple *In-Car* device interfaces can be sketched that

contains buttons, displays, a microphone and a speaker. The next component relates to tasks and provides modelling by demonstration that occurs on the created designs. This automatic modelling relies on ACT-Simple, which in the same way as CogTool translates mental and motor cognitive functions to ACT-R production rules. However Salvucci was concerned with the speed of execution and natively re-implemented a subset of ACT-R and ACT-Simple using C++, which in addition to making the code much simpler to integrate into their system, runs 100 times faster than using the Lisp code. The following component pertains to driving the vehicle where different types of driver age can be selected (as age influences performance). In addition, the steering style can be adjusted and used to represent the aggressiveness of steering control. Another component is the scenarios component that provides a selection of driving environment. Finally, the Results component provides the driver simulation, visualisation and analysis tools. One can inspect the model driver navigating around the chosen environment in real-time where eye movements are positioned on the screen and thus provide feedback on how the interface impacts on the driver's performance. The means to view results are provided in tables or graphical charts.

This research was validated in two experiments. The first experiment is based on phone dialling while driving and attempts to validate Distract-R using results from previous research. The second experiment tests the effect of age on driving performance. For the former experiment, the same tests were used as in section 3.4.3.1. In terms of results Distract-R provides similar results as Salvucci (2001) with the same implications i.e. the latencies related to complete a task for each dialling method are both fairly accurate but are slightly lower than human data. Comparing methods, *Full-voice* requires more time to complete and *Speed-manual* gives the shorter latency in line with earlier experiments (Salvucci 2001). The second experiment relates to the age of the driver when performing multi-tasking while driving i.e. dialling a number when controlling the steering wheel of a car. The premise is that when drivers are driving, age is not significant but when multi-tasking, age does make a difference. The experiments exhibited no major differences when no interactions with other interface features were expected but showed significant changes when drivers were asked to manipulate and dial a cell phone in addition to driving (Salvucci et al. 2005).

Salvucci's work in addition to demonstrating the obvious impact that driving and performing other tasks has on driving performance is also a good platform to test important issues such as multitasking and in particular switching tasks. Since ACT-R does not include a goal stack anymore nor does it have a goal module, Salvucci developed a

*general executive* or goal module, which allows for the addition and removal of goals to and from a set of active goals, leaving this goal module in charge of managing goals and their execution rather than being purely procedural, allowing the switching of goal to be stated in the action of production rules. Since only one goal buffer is present, only one goal can be active and executed at one time. The goal module operates a goal queue based on the principle of first come, first served and each goal is allocated some period of cognitive processing before switching to another goal after a period of time thus incorporating temporal dependence in the goal firing process. The implication is that the general executive can execute the most urgent goal i.e. either "due" or "overdue". Salvucci, for instance uses this extended concept with the experiment mentioned above (Salvucci 2005; Salvucci et al. 2006).

His work provides an insight into different module integration to provide a tool that permits the prototyping of interfaces, with an automatic generation of ACT-R production rules via the ACT-Simple compiler. As running ACT-R in Lisp is much slower than native code, the needed functionality was re-coded natively to allow for real-time experimentation. It also shows how cognitive modelling is becoming a serious contender in predicting human behaviour data that matches human data for a fraction of the cost of both in terms of finance and time.

#### 3.4.3.3 Prediction of energy consumption used by devices relating to interaction

Whilst KLM is used to get timing predictions for user interaction with interfaces, an unexpected use of KLM came from the engineering field in a research called Keystroke-Level Energy Model (KLEM) (Luo and Siewiorek 2007).

As more ubiquitous devices emerge in the market their efficiency in terms of energy consumption is often put into question. Software that is deployed for those highly interactive devices is often not optimised in terms of energy efficiency and run time performance. The main energy efficiency approach is often a global and broad run-time management technique such as aggressive use of idle time etc. The concept behind KLEM was researched to address energy efficiency and its concept is based on prototyping interfaces before the application is implemented and deployed in those devices.

The basic idea behind the research is that more interaction and a broader variety of modalities used to perform a task will lead to more energy being consumed; actions such as tapping buttons etc. Thus less interaction to achieve a task leads to savings in energy. After benchmarking their target device and compiling energy profile for the different

modalities, KLEM uses a user interaction trace (using KLM) to determine the cost in terms of power needed to perform the task, which involves the nature of the interaction and the period it will last. The resulting energy consumption predictions are therefore the sum of all the KLM operator energy and the system activity energy to perform each task i.e. $E_0 = \sum_{s \in S} P_s T_s$ where s is a task primitive, $P_s$ is the power needed to perform this task (from profiles) and $T_s$ is the time needed to perform this operation. CogTool is used to get the needed user interaction through the ACT-R trace provided by CogTool from which all the *MOTOR* functions are retrieved.

Luo and Siewiorek (2007) validated their work successfully using 10 participants whilst monitoring energy consumption of the two used devices. The error between task energy prediction and model energy prediction for one of the devices on average is a significant 4.4% and the other 8.4% (95% confidence interval) which overall provides good accuracy. That project has implications for the research presented in this thesis as it is evidence of a real-time application that uses simulated user models as surrogate users to provide sets of metrics with the difference that these metrics do not focus on usability of the device but instead on the device in operation i.e. optimisation of the energy (Luo and Siewiorek 2007).

#### 3.4.3.4 Predicting user performance: people with physical disabilities

A lot of the research concerned with predicting user performance of computer interfaces is based on skilled user interaction. For users with physical disabilities the range of abilities is more diverse than able-bodied people. As user trials are difficult and expensive (Biswas 2007) is attempting to develop a simulator to help with the evaluation of assistive interfaces. In this research, a task definition and locations of different objects in an interface is used as input, which is then used to predict a cursor trace, likely eye movements and task completion time for the different configuration of the device used.

The Biswas modular system has three main components. A perceptual model inspired by EPIC uses keyboard and mouse events and produces a sequence of eye movements and cursor paths, which are used to predict task completion times. By varying the quality of the image the system predicted visual search for people with or without visual impairment. A cognitive model that simulates users utilises a version of GOMS called Cognitive Perceptual Motor – GOMS (CMP-GOMS) created by (Gray et al. 1993) to simulate the expert users and another model for novices or people with disability inspired by the work of Rieman (Rieman et al. 1996) which involves exploratory learning based on two search spaces; an external space which deals with interface states and an internal space

that deals with knowledge; both of which are mapped to one another. The mapping (i.e. anaphoric reference) in Biswas' project however, is done manually and so is the creation of the user space, which is created by analysis of prior real user interactions. These models use the output of the perceptual model to determine which action to accomplish for the current task. The final component is the motor-behaviour model, which does not use Fitts'law as this is not appropriate for disabled users but instead uses a validated proprietary algorithm based on statistical analysis of cursor traces obtained through observations of user's interaction as well as some amount of trial and error (Biswas & Robinson 2008). At the time of writing this thesis their findings were only confirmed with experimentation using novice able-bodied users and proper testing with their intended target users which is now is due (Biswas et al. 2005; Biswas and Robinson 2007; Biswas and Robinson 2008; Biswas and Robinson 2010).

Their project is relevant to the research presented in this thesis due to the fact it attempts to simulate an interface which is a personalised interface for specific users i.e. in this case for people with physical disabilities or visual impairments.

#### 3.4.3.5 Vision for ACT-R

Much interest has focussed on user interfaces and there has been emphasis on different elements that constitute GUIs to understand and correlate both human behaviour with ACT-R predictions when operating components such as menus, buttons, and text entry controls etc. both in terms of eye and motor movements. The second focus of this experimentation is the validation of the cognitive architecture.

With an earlier version of ACT-R/PM (i.e. which at the time did not include Salvucci's EMMA), Byrne and his colleagues (Byrne et al. 1999) researched this area by simulating these perceptual and motor functions with an ACT-R/PM model, in particular the role of icon searching and saccade movements during menu selection. Overall the experimental data reproduced known behaviours but some discrepancies showed in the raw data collected over a series of experiments. With respect to response times, which predict that it is a function of the target location, their results produced a good fit. With the eye fixations the theory predicts a graded increase of the number of eye fixations within the target location with some visual search strategy based on a top-down approach with some amount of randomness. However, their simulated findings demonstrated different behaviour to where these fixations happened and the one first attended to. With regards to mouse movements, ACT-R/PM model predicts that the mouse movements should follow the eye movements. Their results however show that fewer mouse fixations with the ACT-

R/PM model where needed compared to eye fixations, thus also showing divergence at this level compared to real user behaviour (Byrne et al. 1999; St Amant et al. 2004).

Another experiment of interest is icon search using ACT-R/PM. Fleetwood and Byrne (Fleetwood and Byrne 2001) studied this area and their first set of experiments did not use Movements and Movements of Attention (EMMA), which were used subsequently. The main topic of research was to understand how icons could convey more information without impacting on search time. After gathering real user data to that effect, they constructed two models to illustrate two strategies: a *double-shift (DS)* method whereby the model shift its attention to icons that have common characteristics such as colour with the requested target icon and then shift its attention to the filename below. If the filename coincides with the requested filename, the model shifts its attention to this icon so a procedural action can take place. The other strategy called *text-look (TL)* attempts to focus only on the filename below the icon, and then refocus on the icon in the same manner as DL. The former model does not re-focus on previously seen icons whereas the latter does. These experiments also took into account the icons' complexity. Complex icons or *poor* icons are items that have multiple features that are maybe shared with other icons whereas the *good* quality versions have only one single feature associated between them. The results of their experiments matched the ones obtained with real subjects, that decrease of icon quality affected response time with longer latencies as the quality decreased, and finally the correlation between the DL and TL experiments demonstrated a good fit. The suggestion by the authors was that the number of extra attention shifts generated by the DL approach was cancelled out by the re-visitation of the TL strategy. However, the data also provided some unexplained over-estimation of timings that led to another set of experiments to be conducted with real user to use the eye-tracking techniques to further the understanding of choice of strategies. The new collected data suggested that users were able to identify clusters of icons pre-attentively, by recognising features that were common amongst the group of icons, then moving one from one group of icons to another group while with the poorer quality of icon this behaviour did not occur suggesting the use of better strategy. Moreover, the new data also indicated that for real users TL was more frequently used; as users seemed not to look at the icon itself but directly at the filename in addition to possibly take the length of this filename into account. As a conclusion, they suggested that some underlying assumptions of both the vision module and the conceptual design of their study maybe needed to be improved (Fleetwood and Byrne 2001).

This research was revisited with the ACT-R/PM models using the integrated EMMA system (Salvucci 2000). The experiments remained similar to the ones proposed in the first paper and results vindicated earlier assumptions with regards to grouping of icons and the move from one icon group to another following a nearest strategy as a visual search strategy. The results also found that designing effective icons i.e. their distinctiveness, for a system reduces search times leading to the conclusion that variation of the quality of icons and their number in the display has more influence on search latencies than the strategies used by users, which in itself is an important finding as it provides evidence and re-enforce the case that the design of the icons (always present in today's interface) is an important factor in terms of usability (Fleetwood and Byrne 2003; Fleetwood and Byrne 2006).

The main issue with ACTR is the lack of possible connectivity with applications that were not designed to interact with those architectures. For instance the environment developed for ACT-R runs on the concept of client/server sockets architecture. Both these systems have this functionality implemented into their code. This is not possible with the majority of off the shelve applications that are distributed as precompiled products without any code being distributed; there are exceptions though but the compilation process for a specific platform can be very challenging and time consuming.

As the cognitive community started to focus on HCI issues related to off the shelve products, calls were made to resolve this issue. Segman (St Amant et al. 2005) was created to fill that gap and provides cognitive models with the ability to see the Microsoft Windows graphical interface screen as any real user would and allows ACT-R to interact with Microsoft Windows as if a user was sitting at his desk looking at the computer display, thus providing more direct interaction with the system.

Segman written in Lisp relies on code gluing allowing Segman to connect to native code written in C & C++ exposed in a Dynamic-Link Library (DLL) which among other things provides Segman with means to use low level functionality provided by the Windows Application Programming Interface (API). When loaded, wrapper methods are available to interact with the methods provided and exposed by the DLL. In addition, Segman provides computer vision and lower level image processing to cognitive modellers by processing an image through a process called segmentation. In short, segmentation pre-processes the image using a series of procedures such as reduction of colours, cropping, and translating to name but a few. The next step is the data reduction that uses algorithms to reduce the image data by feature analysis algorithms that can extract pixel groups as regions. This

stage deals mainly with a combination of region growing and shrinking, clustering, boundary detection depending on the outcome that is sought. The final step is the feature extraction phase which extracts, classifies the image features and assigns labels to them (Shah et al. 2003). Therefore, Segman is capable of capturing the desktop screen as a bitmap and through pixel manipulation can create a list of pixel groups and non-overlapping regions of the screen that share the same colour. These pixel groups have features such as number of pixels in the group, its area, height, width, its red, green and blue (RGB) component, its colour in terms of numerical RGB values and its proportion i.e. height divided by its width. These features allow the creation of declarative chunks or patternation. Segman also provides the functionality to combine groups together. For example, a standard embossed button can combine an overall grey colour with highlight in lighter and darker colouring for the embossing effect. Segman can combine these three groups of colour to one overall group. In addition to the feature detection, Segman provides the functionality to initiate commands via the API messaging system to interact with components present in the interface.

Segman is a significant step forward in terms of adding visual capability to ACT-R. It has been tested with applications such as Notepad, or Paint or dialling cell phones (St Amant et al. 2004) and also tested with more serious dynamic applications in terms of complexity such as a simple game of control for which results were very promising as it provides a means for a cognitive model (acting a surrogate user) to see an interface and to interact with it in the same manner as a real user. Another more substantial experiment was performed using a driving game programmed using Java. A system was constructed to conduct this experiment i.e. a Driver User Model in ACT-R and Segman (Van Rooy et al. 2002), its performance that showed lack of robustness in some situations such as strong curvature of the road bend or the inclusion of extra useless procedural knowledge that were set to fire randomly as the model was driving to simulate anxiety as dual task, for instance. The perceptual functionality was based on empirical studies i.e. a dual strategy, where one strategy is used for the detection of the curvature of the road - the region is four degrees below the horizon, the other strategy deals with closer region – the region is seven degrees below the horizon. To provide a balance, between these two strategies, a visual field of 5.5 degrees below the horizon was chosen. When it came to test its effectiveness with more serious games such as a roulette user interface, results were encouraging as it acted a proof of concept but also demonstrated limitations (Shah et al. 2003). Segman has also been tested with SOAR to control some desktop dialling applications. Finally St Amant and his colleagues (St Amant et al. 2005) built a cognitive

model to remote control a simple off the shelve robot ER1 Personal Robot System (Ride 2012). By controlling the robot control centre software interface, Segman was able to control the robot remotely extending to controlling the movement of the robot gripper.

Whilst Segman opens the door to modelling with image processing inclusion, it is restricted in its capacity and the visual process needs to be amended both in terms of cognition accuracy i.e. taking into account human attention allocation and the foveal increase acuity and in terms of its basic object recognition as it stands, objects which are groups of pixel, have to be combined manually (Van Rooy et al. 2002; St Amant et al. 2005; Ritter et al. 2006; Ritter et al. 2007).

Other research that is relevant to this thesis is ACT-CV from Marc Halbrügge (Halbrügge et al. 2007). This research is similar to Segman in some respect as it provides ACT-R with powerful image processing based on the highly developed library called OpenCV. Like Segman, ACT-CV connects to native code written in C & C++ delivered as a linked library, which provides the means to use functionality in the form of native code i.e. C and C++, provided by the library to achieve its visual perception. OpenCV, an open source project backed by Intel and designed for real-time applications, is an optimised multi-platform library containing over 500 hundreds programming functions for real time computer vision (Intel 2010). It is available in the form of libraries depending on the operating system platform and is coded in C and C++. Similar to the algorithm of Segman, OpenCV (OpenCV 2012) is based on the professionally developed concept of object features i.e. group of pixels with some unique attributes. It has all the basic functionality associated with computer vision such as object detection and motion detection (optical flow), it finds groups of pixel that are in motion, find group of pixels that are organised in straight lines and provides face detection using face templates in addition to many low level image processing routines (Gary & Adrian 2008; Gregori 2012). ACT-CV, instead of providing the functionality uses optimised methods provided by OpenCV. This library needs to be loaded by the model. In the past for instance Segman used Allegro Lisp proprietary functionality to load and use the functionality exposed by the DLL. ACT-CV on the other hand does not and uses a combination of Common Foreign Function Interface (CFFI) (Bielman 2006) and Simplified Wrapper and Interface Generator (SWIG) (SWIG 2012) instead thus allowing much more flexibility in terms of running platform. SWIG is an open source library that has a long history since scientists in the Theoretical Physics Division at Los Alamos National Laboratory for building user interfaces to simulation codes first used it. In short, SWIG is a compiler that takes C/C++ declarations exposed in a dynamic library and creates the specific wrappers for a scripting programming platform

needed to access those declarations from other languages that include for example many of the scripting languages such as Lisp and languages such as java. The actual loading of the native code in C++ library from the Lisp platform is done using CFFI for cross platform functionality, which unlike Segman does not rely on the more proprietary functionality of Allegro Lisp.

The experiments carried with ACT-CV are similar to the experiments that were carried with Segman i.e. to demonstrate the publicised impact that driving and performing other tasks at the same time has on driving performance. In this experiment, multitasking tasks is scrutinised in the same manner as Salvucci did with his research mentioned previously. An empirical study was first conducted using a driving simulator where the participants while driving where given an announcement to follow a certain route, then a road sign was displayed after two seconds and the driver has to press specific keys depending on whether the route that was announced was displayed on the road sign either as going straight on, of following a change of course or not displayed at all. The assumption was that the duration of the visual search in those type of tasks is not only influenced by the time to perceive the road sign on the road side but also by the words used for the locations refer to i.e. known or unknown place. Thus to reduce this influence, fictitious places were used in the experiments. Also care was used in terms of not having two places starting with the same letters and the design of the sign was also carefully chosen. An eye tracker was used to track the eye movements during the visual search. In total 19 participants were used, each performing 108 tasks during experimentation. The average time from the presentation of the sign to the participants' reaction was on average 1.9s with only 2% of incorrect response. The results also showed a strong significant correlation between the size and type of the sign and response times. Moreover, the difference of response times only showed when comparing signs of different size but do not with similar type of signs. Furthermore, participants needed 2.7 fixations to attend the sign with 24% of the cases searching the road for the destination mentioned on the sign. An ACT-R model similar to Salvucci's (2002) was created to simulate participants. Analysis of the findings showed that in general ACT-R latencies were shorter with the exceptions where the destination was not displayed on the sign. In addition ACT-R needed more fixations on the sign as was needed by real participants i.e. from 2.7 to 4.1 for the model. The results presented fit well with the current literature i.e. the processing time increases in a linear fashion with the number of target presented in the sign. In term of functionality of the simulation, ACT-CV proved to be a useful tool for processing vision processes. In terms of ACT-R modelling, this experiment showed that it is well possible to design a cognitive model that is capable

of simulating human behaviour despite the shorter latencies that have been reported. Furthermore, this experiment showed that it is well possible to combine this cognitive model with model of driving such as Salvucci. Finally, Halbrügge (2007) reported that the cost in time to create the ACT-R was in the region of days whereas the empirical study took several weeks to complete.

This research further provides evidence that cognitive modelling is useful at many levels, one of which is cost both in terms of time and also in terms of financial costs (Gray et al. 1993). Halbrügge's research is in itself a step forward in terms of adapting vision processes to a cognitive architecture and thus contributing toward a more integrated cognitive architecture (Halbrügge 2007; Halbrügge et al. 2008).

ACT-CV was mentioned in this section for two main reasons. It takes on the experiments done with Segman but uses a well-accepted platform in the industry to simulate the vision processes. Second, ACT-CV combines a series of tools to provide a multi-platform package i.e. through the use of CFFI, SWIG and OpenCV, which is not available with Segman i.e. dedicated to Windows platform only. Thus bearing in mind the use of Open-CV, it presents a more scalable solution both in terms of functionality but also in terms of integration.

In this section, we have looked at some of the prominent research that has been carried out. Most of these examples focus on human behaviour predictive work or providing practical and theoretical enhancements with regard to the cognitive architecture of ACT-R/PM. However, in terms of automatic optimisation of user interface, the literature is non-existent. It is believed that the reason behind the lack of projects is due to the *integration* of ACT-R models within an optimisation engine, which remains unfortunately a major issue. However user interface optimisation has been attempted using some evolution algorithm and some heurists, which unfortunately rely not on human cognition but by other means, which will be discussed in latter sections.

# Chapter 4. Optimisation using evolutionary algorithms

The research presented in this thesis uses an optimisation engine that uses virtual or simulated human cognition and motor functions as part of the assessment of solutions found. The presented research deals with placing components in a user interface in an optimal manner that minimises these functions latencies. It is therefore a placement problem, which is a difficult problem to solve given the fact that many GUI components have different functions that often require different means of interaction. With regards to placing those components on an interface there are many issues that have to be resolved in order to achieve a proper layout. For instance, none of the component can overlap or some components must stay static and not move from their original place to list just a few of the problems to resolve. Given a complicated interface, this search for good layouts can become exhaustive due to the extremely large search space as, not only the components must be properly placed related to one another, but also they must respect constraints too such as spatial and usability constraints. Had this project also included constraints such aesthetics, these would also have to be met with too. In addition, the placement of one component may affect not only the human interaction pertaining to this GUI component and the ones that it directly linked to but also down the line to others as this will often affect the motor functions that resulted from this placement. These constraints can be perceived as costs in a placement problem (Examples of these constraints are discussed in Section 5.9.7).

In its basic form i.e. given a set of components placed in an user interface $C = (1, \dots, c)$, and given a set of locations those components can be located at $L = (1, \dots, l)$, where $l \geq c$ since you must have at least a number of possible locations that is equal or greater to the number of components to place all the components on that interface, the problem is to determine the assignment of components to locations that will result in the minimum cognitive load for interaction between one component and another. Thus the basic form can be expressed as a problem of mapping the one-to-one mapping of the set C into the set L, which results in the minimisation of human cognitive and motor functions.

Thus the feasible region of the problem's solution space contains $n!$ points. For example, with a small interface containing 15 components, the total number of alternatives would be in the region of 1,307,674,368,000; in other words over a trillion possibilities. If one also takes into account costs i.e. component overlapping for instance, which will decreases the number of solutions, that are attached to placements, the possible number of solutions remains extremely large even discounting solutions that are

evidently not feasible and therefore finding a optimal solution to this problem is still a *NP-Hard* problem and the problem often becomes to find a solution that is of *good quality* rather than the definite optimal solution.

Given the large number of possible solutions, and bearing in mind the amount of time to create and test every ones of them and the computing power necessary to compute them, dealing with this type of problems often comes down to creating an algorithm that can move through the solution space to find the best alternatives without attempting to produce them all but to find a solution that produce a good result i.e. an interface that is as optimal as possible at minimising the cognitive and motor functions. The test to decide if a solution is good relies on some sort of metrics which is often returned by a cost function, that becomes larger or smaller depending how good a solution is or vice versa. For instance, in the case of TOISE, the cost function could return a metric that symbolise the amount of time that an interaction scenario takes to achieve in addition to penalties for placing components on the interface either protruding outside the interface surface or overlapping on other components. In this case, a minimisation process would be necessary, as the algorithm should attempt to decrease this latency as well as to decrease penalties.

There two main important branches of search algorithms: i.e. deterministic or not deterministic. The former refers to systematic and less random search processes such as hill-climbing algorithms described in the literature as local search algorithms. A hill-climber often begins with one solution to the problem, which is usually chosen at random. This solution is then mutated and the resulting solution's fitness is then tested to the previous one and if the new fitness is higher than the former one, the old solution is discarded and the new one is then kept as the base solution for the next iteration. The algorithm is then repeated until the mutation process does not produce a sufficient increase in the current fitness otherwise the algorithm returns the current solution. Hill-climbing algorithms are often referred to as *greedy algorithms* because during each of their iteration their aim is to make the best optimal available choice with the aim to find the overall best result. As a consequence, the main issues with greedy algorithms are that they have a tendency to get trapped within local minima or can be affected by poor initial solution choice. Furthermore, in terms of computer processes for the placement problem that we propose, this type of algorithms is computer intensive as they often undergo an exhaustive search of solution space, which can be time consuming.

For the complex minimisation placement problem at hand we need an algorithm that can explore the search space and find a good quality solution quickly without becoming deceived by possible solutions that appear to be good solutions but are in fact short in terms of quality compared to the *global* minimum that may exist in the search space. Deterministic algorithms such as hill-climbers can be useful in instances where the search space in well understood and relatively structured hence allowing special algorithms to take advantage of it, these types of algorithms are advantageous (Forrest and Mitchell 1993; Verhaegh et al. 2006). The other branch of algorithms deals non-deterministic methods, which includes algorithms such as, simulated annealing and genetic algorithms. A short explanation for these methods can be explained as follow. Let's say that a solution space exists which includes a good quality or optimal solution. Thus, an algorithm can eventually discover this solution purely by inspecting this space randomly. Anyhow, this type of random search can often become an exhaustive search in a large solution space, which in terms of processing and operating time is not an option i.e. similar to hill-climbers. However if the search is conducted in a more directed manner i.e. the algorithm jump around from one location of the solution space to another until a relatively potentially good solution is found, then the algorithm can search the vicinity of the surrounding region of that map in the hope of finding a good quality solution nearby. However if the outcome of this local search does not point to a good quality solution, another hop into the other area of the solution space can be performed and the process repeats. These types of algorithms can often significantly outperform classical methods of solution space searches when applied to read-world applications (Fogel 1994; Nolle et al. 2002). In terms of concept, this type of algorithm often gets initialised with a set of solutions that are randomly chosen. A pool of these solutions is then selected that includes some good and poor solutions; the poor solutions are kept in order to allow the algorithm to avoid the chance to be trapped by local minima. The selected solutions are then modified with the hope of producing better quality ones; the amplitude of these modifications decreasing with time or as the overall quality of solutions increase. As the algorithm iterates better solutions emerge to finally converge to a generally good solution though no guaranty is given that the found solution is the absolute best solution.

There two commonly used types of non-deterministic search algorithms - simulated annealing algorithms and genetic algorithms. The former is based on the idea developed by Metropolis (Metropolis et al. 1953; Holland 1975) and later by Kirkpatrick (Kirkpatrick et al. 1983; Mitchell and Forrest 1994) which took the concept of the physical annealing process of solid i.e. known as the thermal process for obtaining low energy states

(equilibrium state) of solid in a heat bath, and applied it to the problem of solving combinatorial optimization problems. In this process the temperature of the heat bath is raised to a maximum melting temperature of the solid so that it is in a liquid state – during which the molecules evolve in a random fashion, which is then followed by a phase where the temperature of the heat bath is carefully lowered until the molecules arrange themselves in a highly structured and stable lattice – the molecule are in the equilibrium state and thus the energy is minimal. If either those stages are not carefully performed, the solid solidifies in a semi-stable state and the energy is not lowered to its true optimum minimum energy state (R. L. Haupt and S. E. Haupt 2004; Verhaegh et al. 2006). The annealing algorithm is an analogy to this process. Moreover it is somewhat similar to the hill-climbing algorithm and thus starts with an initial solution, which is obtained at random from the solution space. During each step of the algorithm, the current solution is perturbed i.e. modified and its fitness is assessed. However unlike the hill-climbing algorithm, if the current solution fitness is lower than the previous one it is not automatically rejected but instead its probability to be retained is calculated depending on the difference of fitness with the previous solution and the current temperature following the equation $p(T) = e^{-\frac{\Delta F}{T}}$ where $p\ (t)$ is the probability of the solution being retained, $\Delta F$ is the difference of fitness between the previous and current solution and $T$ is the current temperature. In order for the annealing algorithm to settle into a thermal equilibrium i.e. to rich a balanced state, each step must be kept to the same temperature for a length of time. Therefore the temperature is decreased in steps following the equation $T_{i+1} = \alpha T_i$ where $T_{i+1}$ is the next temperature, $T_i$ is the current temperature and $\alpha$ is the cooling coefficient. If this cooling time is too short i.e. the cooling coefficient is too low the algorithm is more than likely to be trapped in a local minimum. Too long and the algorithm might not converge to an optimal solution (Kahne 1997; Nolle et al. 2002; Konak et al. 2006)

## 4.1 Genetic Algorithm

One of the most prominent other type of non-deterministic algorithms refers to Genetic Algorithms (GA) (Goldberg & Holland 1988). It is based on the Darwinian theory of evolution. The analogy of this theory and algorithm was first described by Holland in the 1960s (Holland 1975; Huang et al. 2012). Later with his colleagues, he further developed the idea and presented genetic algorithms as an abstraction of this evolution theory which included processes such as mutation, selection and crossover in addition to introduce the notion of schemata (Bäck 1989; Mitchell and Forrest 1994; R. L. Haupt and S. E. Haupt

2004; Jadaan et al. 2008). In short the concept deals with the idea that evolution adapts living organisms to deal with complex environments in which the fittest individuals survive, similarly GA adapts solutions to best solve a problem based upon a fitness factor. The algorithm is based on the fact that strong individuals who have high fitness have the opportunity to pass some of their genetic material i.e. genes to the next generation via reproduction. The individuals who demonstrate poor fitness get discarded over some generations. Over time, during the slow process of evolution, some small mutations can affect individual's genes. If these changes provide additional advantages, the selection process i.e. natural selection will ensure that new individuals bearing those changes evolve from the old ones to eventually replace them to become the dominant species in the population (Koehn 1994; Kahne 1997; Konak et al. 2006). Thus mutation, crossover and selection are the basic processes in a genetic algorithm.

The GA terminology in the literature follows the biological representation is inherits from. A solution is called an individual or a *chromosome*. These chromosomes are composed of discrete units called *genes*, which influences some features of the chromosome. Genes can have many representations such as binary units, integers, float, doubles, strings etc. In genetic algorithms a chromosome often corresponds to a solution in the solution space, which implies that a mapping between the solution space and the chromosomes referred to, as *encoding,* must be implemented. Therefore the GA mechanism operates on this mapping rather than the problem itself. Unlike many algorithms, GA does not operate on a unique solution, but instead operates on a set or *population* of solutions, which is first initialised with random solution. As the search process progresses, the population is checked for its fitness, and from this, a pool of good parents are selected which is used to create the offspring's for the population for the next generation. Finally some individuals from the new population are mutated. A check on the convergence of the population towards the best solution is made and if it is felt that it has converged the GA run exit otherwise it re-iterate, alternatively, the GA can be instructed to run for a specific number of generations. The general process is shown in Figure 24. A chromosome is defined for a particular problem. This is the encoding phase. This population individual has a size i.e. number of elements or dimensions of $N_{nb}$, thus given $p_1, p_2, \dots, p_{nb}$ a chromosome can be represented as an array of these elements i.e. $chromosome = \left[p_1, p_2, \dots, p_{N_{nb}}\right]$. Each chromosome is associated with a cost function to calculate its fitness so for each element of the chromosome the cost is $cost = f(chromosome) = f\left(p_1, p_2, \dots, p_{N_{nb}}\right)$. This is the fitness for this chromosome that will define its selection ranking for matting. Hence for a placement problem such as TOISE,

for a number of components $^{nb}/_{2}$, the number of locations for these components is $^{N_{nb}}/_{2}$, a possible fitness of a solution can be represented as $fitness = f(p_1, p_2, \dots, p_{N_{nb}})$ where $p_1, p_2$ is the location of component $c_1$ in the interface $p_3, p_4$ is the location of component $c_2$, $p_{N_{nb-1}}, p_{N_{nb}}$ is the location of component $c_{N_{nb-1}}$ with $f$ referring to some summation of constraints on overlapping, placement and cognitive and motor function.

Selection is a process in a GA that determines the manner by which it is converges. As shown in Figure 24, genetic algorithms use the selection process to select the individuals from an existing population to insert into a mating pool which are then used by a recombination operator i.e. crossover to generate new offspring that will form the next generation population. It is therefore important that the selected individuals in this pool to have an overall good fitness and the selection process should focus on the selection of those types of individuals in a current population for this mating pool.

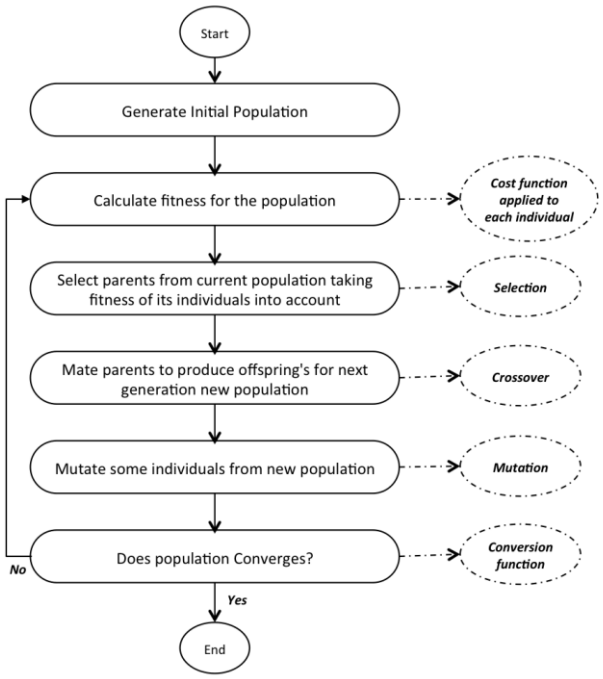



Figure 24: GA Run

The selection pressure is the degree by which the individuals compete with each other so that the fittest individuals are chosen for the matting pool. The higher the selection pressure and the more fit individuals are chosen for the next generation. Consequently, over a number of generations this pressure forces a GA to refine the population in terms of fitness, which ultimately will eventually push the GA to converge to a set of optimal or near-optimal solutions. There are too main issues related to selection pressure. If it is too low, this will lead to an increase of the variability of the diversity of the population and the

GA convergence will be slow and will take unnecessary longer to find an optimal solution, adding extra computer processing time, often stalling the GA from settling down. Conversely, if it is too high, it will decrease the population diversity and the GA might be driven to converge quickly and to a local sub-optimal solution, which could be far of the optimal solution. In other words, the GA settles down too fast (Mitchell 1996; Huang et al. 2012). Selection schemes come in two major flavours i.e. proportionate and ordinal based selection. The former bases its selection on the fitness of the individuals relative to the fitness of the entire population. The latter base its selection on the rank of individuals compared to the other individuals ranked according to the fitness in the population. Therefore the main difference is that the selection pressure is directly proportional to the fitness of individuals for the former whereas it is not for the latter i.e. it is only based on the ordering of individuals of a population. Examples of commonly used proportionate selections are *proportionate* selection, *stochastic remainder* selection and *stochastic universal* selection and examples of *ordinal-based* selections are *tournament* selection, *truncation* selection and *linear ranking* selection. It follows that the selection operator is a critical component in delivering optimal solution when using GA (Bäck 1989; Reeves 2003). It is not the place in this work to describe in details all the selection algorithms that are used today, as the list is exhaustive. Much research has been undertaken in this field. We will simply briefly introduce some to provide information regarding their features however. The classical *Fitness Proportional* Selection (FPS) also known as the *Roulette Wheel Selection* (RWS) is based on the simple algorithm where the expected number of copies for a particular individual is based on its fitness compared to the population fitness (which normalises it) usually using a roulette wheel to make the selection. In this process each individual is assigned a slice of the roulette wheel with the size of the slice being proportional to the individual's fitness related to the population fitness. A random number is generated from $r = (0,1)$. Starting at the beginning of the wheel, the cumulative probability is summed until it is equal or greater to the $r$ value. The entire circumference of the wheel equals the sum of all individual fitness's of the population individuals (Hancock 1994; Jadaan et al. 2008). The individual under the wheel selector is then selected for the mating pool. The process is repeated for the number of individuals that are required for the mating pool. This algorithm however has reported issues associated with it (Goldberg and Deb 1991; Koehn 1994; Miller and Goldberg 1996). This process leads to early convergence, as at the start of the GA run, there are many individuals with poor fitness and just some much fitter. Since the selection is purely based on stochastic selection i.e. random, poor individuals can be selected multiple times with only a small proportion of fitter ones entering the matting pool. Consequently toward

the end of the run the fitness of the individuals and the population overall has little variation. In addition, as poorer individuals multiply rapidly, this stops the GA explorative search leading often to early convergence to local optimal solution with the few fitter ones driving the GA to this early convergence (Mitchell 1996; Luke and Spector 2010). An attempt to avoid this rapid degradation of the population fitness is to omit replacing individuals into the selection pool once they are selected. Whist helping, this does not provide a robust solution. A proposed solution to this problem is to use *Stochastic Universal* Selection (SUS). This type of selection is similar to the roulette wheel but rather than using the cumulative fitness probability as the mean of selecting individuals, selections points are distributed on regular interval around the roulette wheel once an initial pointer is placed at random. The selected individuals are the ones that fall under those pointers. This provides a chance to fit individuals to have many copies entering the matting pool. This partially addresses the problems mentioned previously but this selection type is still susceptible to early convergence. In addition, this does not solve the issue of scaling when for instance using real values strings as part of the encoding (Hancock 1994; Miller and Goldberg 1995; Wall 1998; Reeves 2003; Sivaraj and Ravichandran 2011). Another type of selection is *ordinal* selection based on ranking. One of the major attributes of ranking selection is that it is not directly proportional to fitness and is normally the preferred alternative to proportional selection mainly because of the scaling problems and early convergence (Goldberg and Deb 1991; Miller and Goldberg 1996; Harik et al. 1999). A selection often referred to is the *Tournament* Selection (TS). This type of selection is based on ranking the parents in terms of fitness and running a tournament contest using a small group of these individuals randomly selected with the individual with the fittest selected for mating. This type of selection provides a robust mechanism in which the selective pressure varies proportionally with the number of individuals that are present during the tournament i.e. more participants increases this pressure but in the literature this value is often set to two (Thierens 2002; Luke and Spector 2010). Because of its nature, it is often used with noisy fitness function (Spears 1993; Miller and Goldberg 1995).

To put it in perspective with the research presented in this thesis, when dealing with human visual perception, which has a tolerance set by default to 0.5 degree of visual angle, the fitness function would report similar values for many layout solutions that are slightly different. Because of its elitist behaviour and competitive nature, the tournament selection often selects the solutions in a manner that favours the fittest and typically chooses more valued individuals than would proportional selection otherwise. It must be noted that this

selection uses RWS to select the participants at random (Tate and Smith 1998; Wall 1998; Sivaraj and Ravichandran 2011).

Other important genetic operators are the crossover and mutation operators. The crossover deals with mating of two selected parents to create one or more offspring and is often associated with the exploitation of the solution space for a given problem. The most common reproduction form involves two parents producing two offspring. In this instance some points $p = 1 \ldots n$ i.e. kinetochores are randomly selected between the first and last gene of the parent chromosome. The gene information from both parents is then copied onto both the offspring chromosome with respect to those points. Thus the two offspring inherit both parents genetic material. There are many forms of crossovers that have been studied but all have the same idea behind which is to recombine building blocks on different chromosomes. There are known issues with implementing crossovers. One well known issue is hitchhiking and comes with the one-point crossover whereby given only the one point, poor genetic material can tag alone good genes over many generations. The other problem with this style of crossover is that the genes at the beginning and end of the chromosome will often been replicated. Therefore many projects use the multi-point style of crossover to overcome these issues. One crossover that is worth mentioning despite its potential disruptive effect is the uniform crossover, which has the potential to disturb any genes present in a chromosome. The choice of the crossover method used in a GA is often related to the type of other operators used i.e. fitness function, the encoding used and mutation amongst other aspects. However it all depends on the type of problem to solve and therefore the encoding used. In layout or placement problems it is often the case that special crossover operator are created to deal with the special type of encoding. These issues will be discussed in the next section when examples of placement projects will be discussed in more details.

Another important operator is the mutation operator, which is used, in genetic algorithm to maintain genetic diversity in a population from one generation to the next. This operator alters one or more genes in a chromosome in an arbitrary manner thus introducing the extra variability in a population that is often needed in genetic algorithms. Mutation is often perceived as the second tool in a GA to explore the solution space in addition to its exploration role. However, whilst crossover guaranties preservation of genetic material, mutation does not. There is however proprietary GAs that does not use the mutation operator. Compact Genetic Algorithm (cGA) is one of these GA that has been successful in standard tests that only uses a uniform crossover with a selection pressure provided by a tournament selection with replacement (Harik et al. 1999; Hooda 2012).

Both the crossover and mutation operators are assumed to have rate associated with them that provides the probability that changes will occur. Finding the right rates for these operators can become a tedious job. For instance Thierens in (Thierens 2002; Perrins 2008) discusses several adaptive mutation controls that can be used to eliminate the need for a static mutation parameter. These attempts are based on declining the mutation rate over a GA run with binary encoding however. Nonetheless, in the literature, there seem to have a standard for those rates (which off course are dependent of the problem to solve). The crossover rate is often thought to be $0.6 \leq r_c \leq 0.7$ where as the rate for mutation is in the region of 0.001. Moreover, the literature often attempts to question the relevance of these operators (Spears 1993; Maveryx 2011), which are at times is misleading. In many cases the consensus is that the driving force behind GA is its selection of parents for breeding, new generation based on their fitness and the discarding of the poor individuals as the main factors for contributing to the general increase of the population fitness. Whereas as crossover or mutation has had a major contribution is somewhat secondary as above all the selection pressure acts on the population individuals regardless if they are born through breeding or brought about by mutation. These operators however as important they are help to raise the individuals' fitness but without a strong and appropriate selection mechanism, these efforts will not translate to a fitter population (Tate and Smith 1998).

## 4.2 Application of Genetic Algorithm in the domain of component placement

In the realm of applications dealing with component placement on 2D or 3D environments, the research done in this field has increased over the years. There are however leading fields which lead or steer the research towards helping professional in area such as electronics i.e. System-on-Chip (SoC) (Manikas and Cain 1996; Lohn 2002; FriedMan 2009) , Field-Programmable Gate Array (FPGA) (Ando and Iba 2000), construction site (Jang et al. 2003), building (Yang et al. 2002; Narahara and Terzidis 2007; Klanac et al. 2008) or plant layout (Gero 1999; Balakrishnan and Cheng 2000; Osman et al. 2003; Jagielski and Gero 2009), ship building (Thapatsuwan et al. 2007), factory layout (Ajenblit and Wainwright 1998; Pérez et al. 2005; Ho and Perng 2009), document assembly (Purvis 2009) and some dealing with User interfaces (Oliver et al. 2002; Brewbaker 2008) to name but a few. The research and work is this field is far too sizable to provide a comprehensive overview therefore this thesis will attempt to describe some work that concentrates or relates best to the work this thesis presents in this thesis such as discussed by Brewbaker and Oliver.

The Brewbaker's research is related to the research within this thesis as it attempts to optimise some user interaction virtual keyboards, using a form Shannon Fitt's law (which is used in ACT-R).

Its main focus is to use a genetic algorithm approach to optimise virtual keyboards layouts, namely QUERTY and FITALY, the former being a keyboard optimised for stylus use whereas the former is a ubiquitous keyboard found in many keyboards. The optimisation process focuses on two target users referring to two types of typists: English language typists and C programmers using Shakespeare literature for the former and the Linux kernel code for the latest which are used to test the system.

The fitness function for the GA is based on Shannon version of Fitt's law, which has been modified to take in the Euclidian distance between the keyboard keys when typing from one key to another. The GA encoding encodes each key for a character using unique real numbers in a matrix representing all the key of the keyboard i.e. (1…number of keys). The roulette selection is used to provide better population diversity for the selection and is using a *two points* crossover.

Brewbaker succeeded at evolving QUERTY keyboards that displays certain particular traits such as placing OTAIE in the home row of the keyboard for the evolved Shakespeare test which fits well with the English language as most of those letters i.e. AIEO are frequently used. For the programming tests, sequences of letters or words such as FOR, N, I were located in the home row which fits well for this type of programming language as they are often used. Tests for the FITALY keyboards were also successful are putting close to each other letters commonly used in the Shakespeare test i.e. A, I, O and U but placed them diagonally around the middle of the keyboard. Tests for the same keyboard focusing on C programming placed letters N, I and A in the middle, with the word PRINT nearly contiguous. This again fits well with the Linux kernel code which includes many debug code lines using this word.

Brewbaker compared also this method with a random approach. His results showed that the random method came last in terms of timing performance returned by the Fitt's law based fitness function, with the GA evolved keyboards being the best high performance solutions, the standard QUERTY being in the middle in terms of performance.

The main issue with the work presented in this paper is that it does not take into account any movement preparation, execution latencies but only focuses on expert user fast movements. However, it shows potential of using GAs for optimising user interfaces

specific applications using a fitness function that takes into account a basic human performance when operating an interface.

Oliver's research that is presented in his paper deals with automatic optimisation of web pages layout, which is its relevance to the research presented in this thesis. What is novel in this project is that it is based on an IGA which uses actual user feedback as fitness assessment for the system proposed layouts. Its main purpose is to suggest to users different visually optimised layouts based on some user specification (font, size, colours etc.) and therefore eliminate technical knowledge that is often required for such task.

This work focuses on two parts: one that deals with individual styles of the components and the other of the overall layout of the web page, where users can optimise either one. To help with the usability and ease of use, only 12 layouts are proposed to the users at one time.

Mutation rate controls the type of layout that is suggested by controlling their diversity in terms of component placement. The run starts with a high mutation rate, which decreases over time, thus suggesting a diversity of layouts that gradually focuses down to the user preferences. Some controls were implemented to ensure readability of the text and their luminance. Any failure results in the run re-starting. The crossover is based on a uniform crossover (with probability rate $= 0.5$).

The layout is based on a HTML table where all the objects are contained in the table's cells, which can be merged together hence allowing for an increase variety of layout representations. The fitness of the representation takes into account two constraints: overlapping and the inclusion of individual object in one cell. The mutation operator assigns one object to a cell at random.

The study reported by Oliver is in operation and represents a real world problem where scalability is paramount that has shown itself successful in a real life business situation based on a GA to provide optimised layouts with direct real user input as part of its fitness assessment.

# Chapter 5. Software Design - TOISE

## 5.1 Introduction

The main part of the concept discussed in this thesis relies extensively on ACTR modelling. In the first sub-section of this chapter I will discuss an overview of the road taken to develop the concept and in the later chapters I will discuss the ways the system was implemented. Finishing this section, I will discuss the points that led to the development of an updated version, namely the version of TOISE compiled from C++ code.

Looking at ways users interact with software, it is possible to define primitive actions that are taken to interact with a system.

- Typing
- Clicking
- Dragging
- Moving limb

Those basic actions are often mixed together as a scenario to perform a goal. For instance, if one has to click on a specific series of buttons to achieve a task, one has to move the hand in position, click the component, then move the hand to another location, maybe click on that particular component, then perform some typing, then move the hand to another location and maybe drag an object to some location. Regardless of the complexity of the task, the task can often be divided into a series of smaller sub-tasks, which themselves can be split over a series of sub-actions to perform if necessary. In this thesis, we name those subtasks *primitive actions*. We named them *primitives* because they cannot be sub-divided into smaller actions.

In order to obtain this type of data, an action recorder must record user actions when instrumenting a software prototype; this recorder must be able to return these interaction primitives. The following section will describe the software that allows this type of recording. It is called in this project: *Robot*.

Once a model of user interaction has been created, this interaction must be simulated using the ACT-R cognitive architecture. Hence, models must be created to simulate these action primitives with its supporting software. In the following section, this side of the project will be discussed. Finally, an optimisation system using these models is used as

part of a genetic algorithm. The final part of this chapter discusses the software that optimises layouts and presents an optimal setup for the layout.

TOISE uses both perceptual and the motor modules to interact with a virtual world named the *device* represented in ACT-R/PM by the device module. The device simulates a computer (but can simulate other environments if needed) that is useful for HCI research and TOISE. Visual elements such as objects and object locations can be added to the device. The code fragment shown below for instance creates two chunks; one for an object and one for its visual location and add them to the device interface, which is then selected as the default visual interface for the current model.

```
...
(push (car (define-chunks-fct `((
   VISUAL-A-LOCATION
   isa OBJECT-A-LOCATION
   color      black
   value      OBJECT-A
   kind       OBJECT-A
   screen-x   ,obj1X
   screen-y   ,obj1Y
   width      ,obj1W
   height     ,obj1H)))) visual-location-chunks )

 (push (define-chunks-fct `((
   A-OBJECT
   isa        OBJECT-A
   CONTENT    "object1"
   value      "object1"
   width      ,obj1W
   height     ,obj1H))) visual-object-chunks )

(let((the-device (pairlis visual-location-chunks visual-object-chunks)))

(install-device the-devic
```

Once the device is selected, the vision module provides the functionality to interact with this interface as well as to provide information about what can be seen in it. It is worth mentioning that ACT-R/PM does not model eyes movements. This functionality is external to the core system and is provided with Eye Movements and Movements of Attention (EMMA) created by Salvucci. ACT-R can be set to force EMMA to replace its standard vision module.

As will be discuss further down, TOISE decomposes a scenario of actions to its basic primitives; the buffers that it needs to interact with are therefore limited and are displayed below. First it is goal directed which means that there is a need to change memory content to perform different tasks. Thus memory needs to be retrieved from declarative memory, location of objects needs to be found and attended to, the visual module needs to be issued commands to direct the eyes and finally the motor module needs to perform primitive actions.

| Buffer | Module | Commands |
|---|---|---|
| **Goal** | Framework i.e. cognition | Changes or replace the content of memory element |
| **Retrieval** | Framework i.e. cognition | The main action is the retrieval from the declarative memory of chunks with the highest activation that fulfil all the constraints mentioned in the LHS |
| **Visual-location** | Vision | This command moves the system visual attention to a specified location. |
| **Manual** | Motor | Execute actions such moving mouse, clicking a mouse etc. |

## 5.2 Recording Interaction

The software that was constructing to record the user interaction is called Robot. The main specification for Robot was four folds. Firstly, Robot should record all the user interactions and secondly create a model for that interaction scenario. This model can then be imported as a scenario of interaction that can be simulated with a series of ACT-R models. Thirdly, the recorder should generate a model of the interface that was used during the interaction, which can be used by the optimisation engine as a virtual interface that provides all the metrics necessary during the process. Finally, an extra specification was for the software to be able to re-run the interaction on the specific software. See Figure 25.

### 5.2.1 Alternatives

Testing and test automation is a vast domain which is often perceived as the most cost effective method for testing software products. There are two main general approaches to test automation. Code driven testing that is often based on frameworks that interface classes and backend code, which are validated by input arguments. The other kind which is part of our domain of interest are graphical user interface testing frameworks that generate system events such as mouse and keyboard input which exercise the underlying code and validate user interface behaviour. These tools are often expensive and often require knowledge of the application under test often relying of professionals to perform the necessary steps (Hooda 2012).


Figure 25: Interaction Recording[31]

One of the earliest effort in producing some user input automation was made by Sun Microsystems when integrating the java Robot class as part of the 1.3 Java JDK in 2000 (Ginbayashi 2004). It is however difficult to use this class as exact interface object metrics must be precisely known to generate events properly. This class is often used as a base class in many frameworks such as Abbot, which will be discussed further on this section. For this reasons, a few attempts were made using Java Robot but difficulties obtaining consistent data to model the interaction and software interface were experienced, and which led to a change of direction.

The idea behind this change was to use a free Java testing framework, apply some code changes to it to transform the testing platform into a user interaction recorder. Comprehensive testing suites are often frameworks that are integrated into programming environment to instrument the code and offer an automated functional testing tool as well as a regression-testing tool. Others are not but rely on bootstrap code to load some test scripts alongside the application code to be tested, which often allows the framework to load other needed testing modules. Two of these free public domain sophisticated frameworks are the well-known Dojo[32] (Perrins 2008) (See Figure 26), endorsed by IBM[33] and Maveryx (Maveryx 2011) (See Figure 27).

[31] Courtesy of: http://bfmexotic.com/wp-content/uploads/2011/08/computer-based-test.jpg

[32] http://dojotoolkit.org/

[33] http://www.ibm.com/developerworks/web/library/wa-aj-doh/index.html

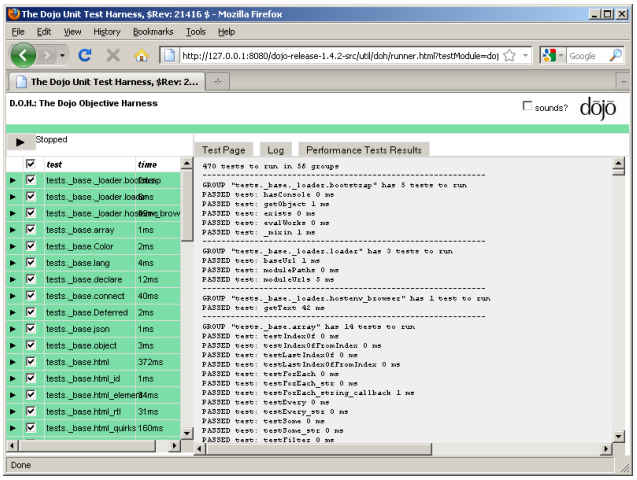



Figure 26: Dojo testing suite[34]

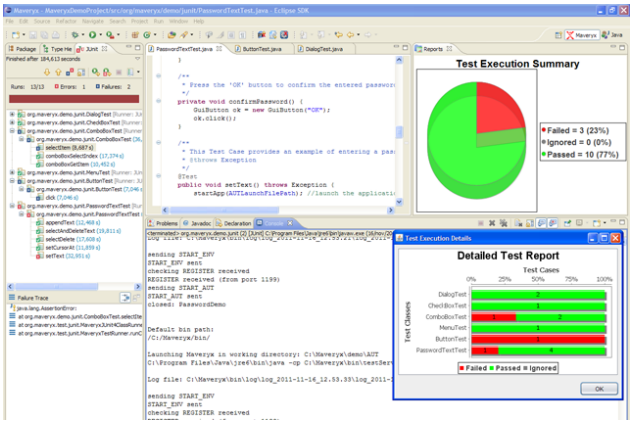



Figure 27: Maveryx Testing Suite[35]

However, the idea of bootstrapping / integrating some bespoke code onto an existing application source code renders the goal of user interaction recording much more difficult and less intuitive, not to say attractive as in most cases the original source code is not accessible. In addition, those innovative frameworks are based on identifying UI elements that are present in interfaces and needed by testing scripts, rather than having the code being instrumented by demonstration to discover the elements that are needed for a specific task.

There is one of these frameworks that differ from the others mainly because it is by definition a framework that focuses on testing Java GUI's rather than testing the underlined operational code. Rather than relying on additional testing scripting code, it dynamically loads an application, creates a model of the application interface and records

[34] http://blog.siliconforks.com/wp-content/uploads/2010/05/jscoverage-dojo.png

[35] http://marketplace.eclipse.org/nominations/product/maveryx

the interaction using a set of semantic recorders that are hooked into all the elements of a Java software interface. This framework is called Abbot (Wall 2011)[36]. It is built upon the *java.awt.Robot* class to provide an automated event generation and validation framework for Swing GUI components that are present in java applications as well as and functional testing. The suite contains also The Costello script editor that can record user actions and facilitate script construction and maintenance (See Figure 28 and Table 1). This GUI regression testing toolkit is reliable and despite the fact that it has some glitches (Ames and Jie 2004).

Table 1: Benefits of Abbot Library

| **Benefit** | **Explanation** |
|---|---|
| **Reliable reproduction of user input** | The reason that often graphical user interface do not get the required amount of testing is often because it is usually not a simple task to simulate user input. Abbot solves this issue by making this process seamless. |
| **Scripted control of actions and inspection** | Rather than having to write extra code for testing, the test scripts are dynamically interpreted by Abbot |
| **Loose component bindings** | User interface tend to change over time. Thus a raw event recorder and playback is more likely to break during the development of a product, as the component absolute coordinates tend to change with it. Abbot does not rely on component location as long as it can find them dynamically |
| **Specify high-level semantic actions whist using low-level Operating System events to implement those action** | As already discuss, the *java.awt.Robot* is capable of simulating user events but is too low level to be used easily. Abbot has a built-in level of abstraction on top of the java Robot in the same way Swing components present a layer of abstraction to related Awt components thus making testing easier to construct. |
| **Support recording and editing of high-level semantic events** | In many circumstances, test scripts are written as XML scripts (often by hand), Abbot comes with its own test script management editor that let user record, edit and manage test scripts i.e. Costello |

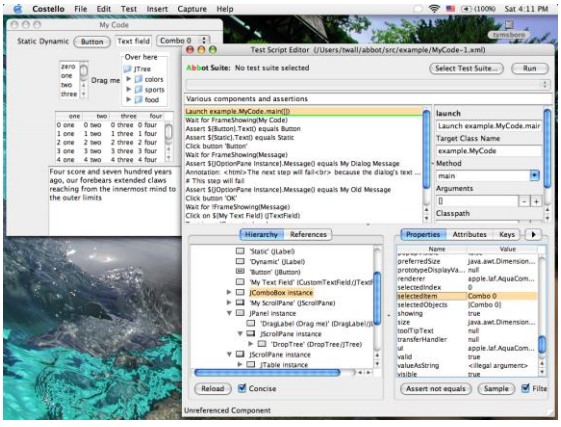


Figure 28: Abbot Testing Suite[37]

### 5.2.2 Robot based on Abbot

The Abbot Framework meets many of the requirements that were sought, but being a testing platform, its main focus is on processing user actions on an interface and functional testing rather than modelling it as user interaction. Thus, a few code adjustments were

[36] Reference: http://abbot.sourceforge.net/doc/overview.shtml

[37] http://abbot.sourceforge.net/doc/images/costello.png

necessary to transform this platform to meet those requirements. Moreover, in order to create a model of the interface, parental inheritance has to be properly resolved. This was necessary as Java applications often rely on parental metrics to set the components in the right locations on the GUI; functionality unneeded in testing. Other necessary addition dealt with timing. In testing mode domain, these types of metrics are not needed and thus are not included, but they are essential if proper user interaction needs to be recorded. Underlying code was therefore included to expand the Abbot functionality to record timings in milliseconds. Further additions in the recorders were also necessary to record location metrics in the model (again not necessary when testing in the domain for which Abbot was created). Those changes had to filter out to the Costello Editor, which is the main manager for the scripting and parsing functionality. The main issue with Abbot was discovered when dragging operations took place. Java (unlike other platforms) generates an Abstract Window Toolkit (AWT) Event when this type of action is performed. Unlike the Windows API which generates an API event when the drag action is first performed when selecting an object to drag which refers to the object that is being dragged and then another message when the selected object is dropped relating to the object that becomes the container for that object, Java does not and the drop message still refer to the object being dragged, not referring to the new container that receives that object. To create a user interaction model, one needs to know the object that is being selected in addition to the object that will contain that object in the future. In a testing environment, this is not an issue but it becomes one when trying to analyse user interaction and model a user interface.

In order to deal with this issue, the need to investigate the messages coming directly from the API was necessary rather than relying on the Awt event message pump for the drag and drop events received from Java. The only way to achieve this was to build a Dynamic Linked Library (DLL) with the necessary code to retrieve events directly from the API, and dynamically loading this library into the Abbot Suite backend java code, replacing the code dealing with this type of recording. To achieve this SWIG[38] (SWIG 2012) was used to wrap the code contained into the library into the Java code.

SWIG is a compiler that wraps an interface around a native platform of choice, whose code can be compiled in a native form. Unlike some other alternatives i.e. Java Remote Method Invocation (RMI) based on a server / client architecture where the former

38 More Information at: http://www.swig.org/index.php

instantiate remote objects and make them available to client applications which can obtain remote reference of those objects and invoke methods exported by the server, using object serialization to marshal and un-marshal parameters, SWIG does not produce stubs i.e. proxy interface to remote objects, but produces code that can be compiled into the java code and ran, thus avoiding the difficulties of writing server/client code to run some functionality. SWIG is mature system that was first released in July 1995 by Dave Beazley (Theoretical Physics Division at Los Alamos National Laboratory). The main advantage is that the wrapping is automatic, can be easily implemented and is scalable (Langtangen 2010). SWIG has been used in projects like SubVersion, Web-based Analysis and Visualization Environment (Weave)[39] (Dufilie et al. 2012), Python Open-Oriented Graphics Rendering Engine,(PyOgre)[40] (Dawson 2010) (a Python version of Ogre), Shogun[41] (Sonnenburg et al. 2010) and OpenCV (Thompson & Stevenson 2011) for which SWIG is used to interface platforms such as Python. It is support at the time of writing eighteen platforms including Java, Allegro Lisp and CFFI.

The change of focus from a testing application to a user interaction-recording suite can be seen in Figure 29. Figure 30 displays the models generated by the software with its two parts: one that deals with the interface whilst the other focuses on the user interaction. In the figure the model lines (in red) correspond to the modelling that pertains to the interface, which includes all the components that took part in the interactions recording. It is a sub-model that TOISE will parse to create his own virtual interface description. The green section represents the second sub-models that correspond to the type of actions that were performed during the interaction i.e. clicking, dragging or typing etc. TOISE parse this model (in green) and will link these actions to the objects (in red) that were included during these specific tasks.

Figure 31 and Figure 32 display specific details of the type of information that model the interface and the user interaction. The details provided are sufficient to model many aspects of the application and the user interaction provided that care is taken to reconstruct the GUI given parental inheritance when parsing these models to reconstruct virtually both the application interface and the interaction that took place on that interface during the test.

---

[39]More information at http://www.oicweave.org/ & https://github.com/IVPR/Weave

[40] More Information at http://www.ogre3d.org/tikiwiki/PyOgre

[41] More Information at http://www.shogun-toolbox.org/doc/en/2.0.1/index.html

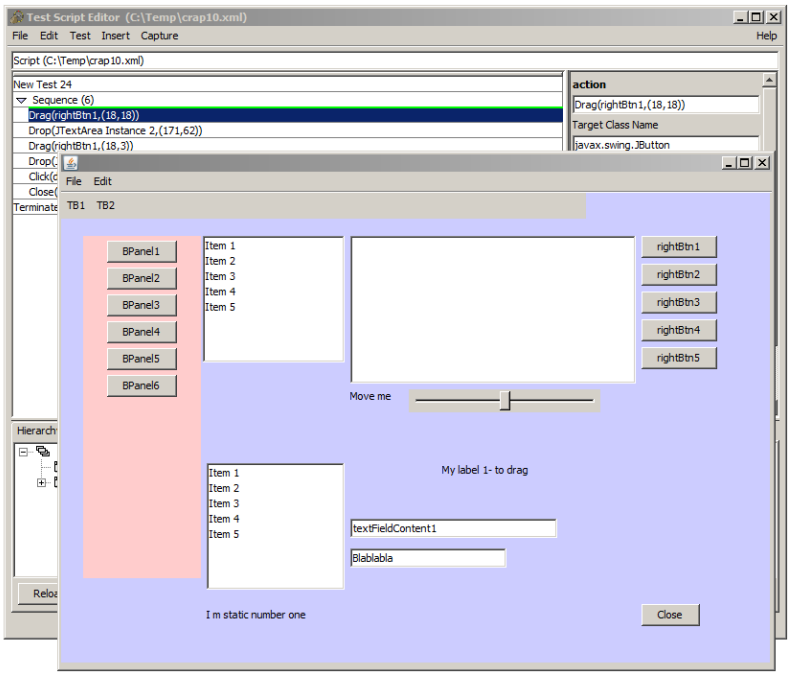



Figure 29: Robot after recording a Drag and Drop action


Figure 30: Robot Interface and Interaction model

```
<component
 baseClass="class javax.swing.AbstractButton"
 class="javax.swing.JButton"
 height="23"
 id="Automatic Recording"
 index="0"
 parent="jPanel3"
 text="Automatic Recording"
 width="131"
 window="dragANDdropFrame Instance"
 x="26"
 y="43"
 />
```

Figure 31: Portion of a GUI models generated by Robot

The sequence of action can be seen Figure 33. In this screenshot, one can see that each sub-task can be edited or tweaked as needed either using Costello or by hand.

```
<action
  args="rightBtn1,(18%2c18)"
  class="javax.swing.JButton"
  method="actionDrag"
  refTiming="1350138040737"
  timing="1350138042409"
  x="18"
  y="18"
  />
```

Figure 32: Portion of the user interaction generated by Robot

The information that is shown is retrieved from the application as it is dynamically loaded and then completed when a user interaction has taken place. In addition more sequences of interaction can be added to existing or new sequence. Therefore many complex sets of tasks can be recorded, run and edited with Robot based on the Abbot framework.

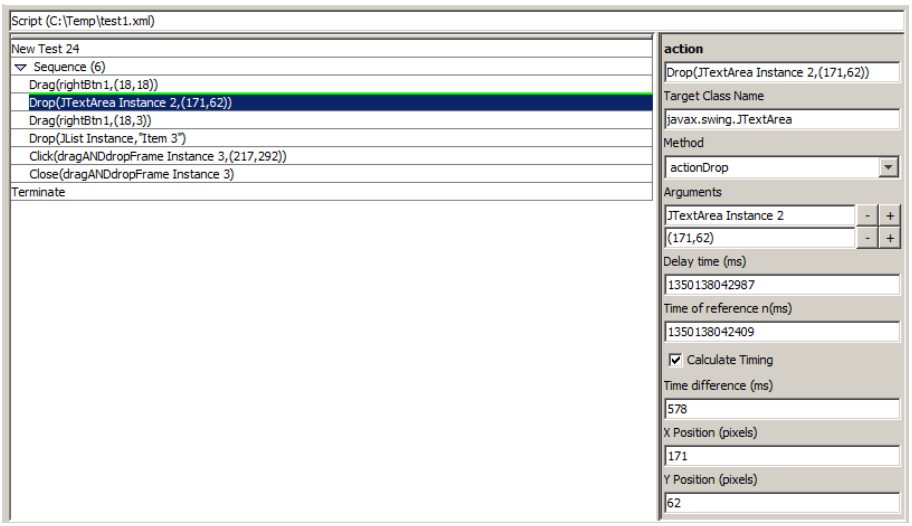



Figure 33: Recording of a sequence of action

Moreover, Abbot had the base code to record motion. This functionality was extended in the same manner as the action recording and is available to further investigate user interaction on the motion level as seen in Figure 34. Furthermore, another issue that was dealt with was to retrieve the real containers of the interface objects rather than containing abstract class objects.

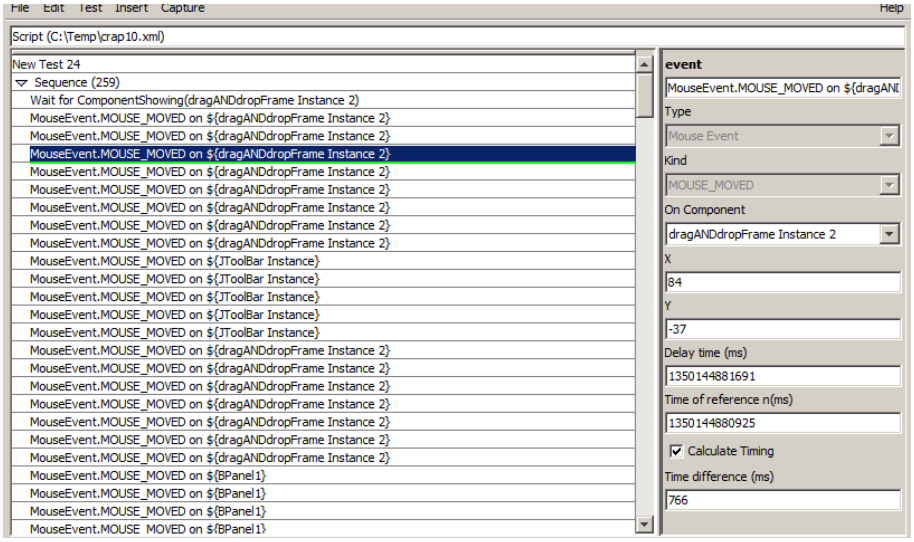



Figure 34: Robot extended motion captured

## 5.3 Modelling Primitives

In the project that the thesis presents, modelling ACT-R primitives actions were done starting from a plausible GOMS model as shown in Figure 35, which is the primitive that will be used in this section to demonstrate how the actual ACT-R models were obtained.

```
Define_model: "JCG Dragging Example"
Starting_goal is DragDrop1 Action.

Visual_object: SW1_Object
  Content is "Object1".

Visual_object: SW2_Object
   Content is "Object2".

Method_for_goal: DragDrop1 Action
    Step 2. Look_for_object_whose Content is "Object1" and_store_under <src-target>.
    Step 3. Point_to <src-target>; Delete <src-target>.
    Step 4. Hold_down mouse_button.
    Step 5. Look_for_object_whose Content is "Object2" and_store_under <dest-target>.
    Step 6. Point_to <dest-target>; Delete <dest-target>.
    Step 7. Release mouse_button.
```

Figure 35: Drag & Drop GOMS model

G2A[42] is a runnable system created by St Amant et al. that automatically abstracts ACT-R productions from more abstract and high level GOMSL models (St-Amant et al. 2005). Using a GOMS model as a base, G2A parses this model and generates the ACT-R productions rules by mapping the GOMS operators (actions to be performed) to ACT-R productions rules which are integrated into an ACT-R model with predicted timings within 5% of GOMS models latencies (St-Amant and Ritter 2004; Byrne 2005; Karray et al. 2008).

The only issue with this solution is its reliance on ACT-R version 5. However, as we are dealing with simple primitives in the research proposed in this thesis, this does not pose a problem to the conversion to an ACT-R model suitable with version 6.0. The resulting ACT-

[42]G2A is available at http://www4.ncsu.edu/~stamant/G2A/

R model just needs to be modified and tweaked to reflect the changes necessary for ACT-R 6 to take care for instance of Visual State and Manual State which are no longer used in ACT-R 6. The available G2A generated ACT-R model can be perused in Appendix A with a sample run displayed in Appendix B. This model gives us the main mainframe from which a final model can be derived and modified as a macro to allow for the encapsulation of the ACT-R model, which can then repeatedly invoked.

## 5.4 Decomposition a series of task

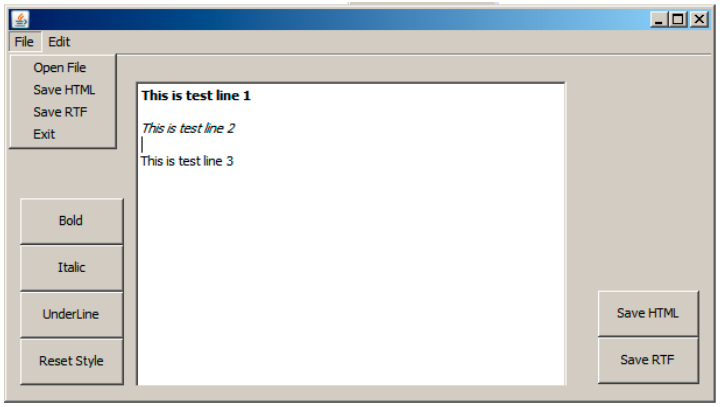



Figure 36: Test Application small text editor in Java

Any task involving a series of action on an interface can be decomposed into a series of independent tasks.

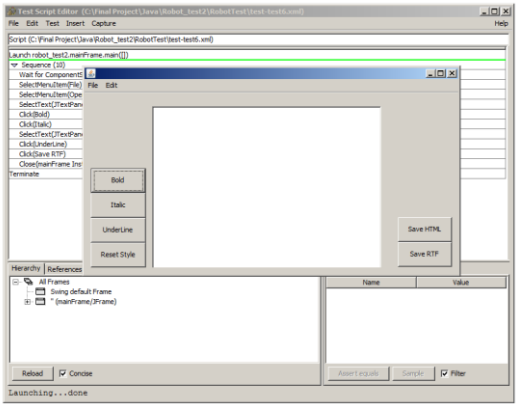


Figure 37: Demo Test Application exercised using Robot

Looking at Figure 36, and Figure 37, the test consists simply of entering the text pane, adding three lines separated by two empty lines, formatting the first line in Bold, formatting the second line in Italic, formatting the last line using the Underline, saving this as a RTF file and finally exiting the application using the Exit menu item.

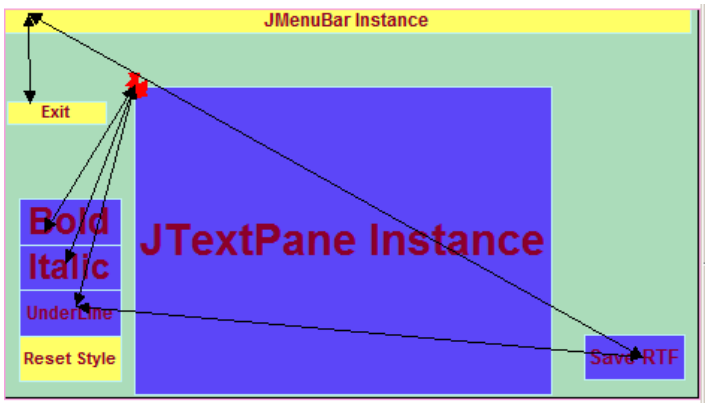



Figure 38: Small test

The Robot model shown in Figure 38 can be described as a scenario:

- Click(JTextPane Instance,0%@%113-34)
- KeyString(JTextPane Instance,This is test line 1)
- ActionMap(JTextPane Instance,insert-break)
- ActionMap(JTextPane Instance,insert-break)
- KeyString(JTextPane Instance,This is test line 2)
- ActionMap(JTextPane Instance,insert-break)
- ActionMap(JTextPane Instance,insert-break)
- KeyString(JTextPane Instance,This is test line 3)
- ActionMap(JTextPane Instance,insert-break)
- SelectText(JTextPane Instance,19,0%@%91-10:-12-5)
- Click(Bold)
- SelectText(JTextPane Instance,40,21%@%108-43:-4-44)
- Click(Italic)
- SelectText(JTextPane Instance,61,42%@%89-61:-1-64)
- Click(UnderLine)
- Click(Save RTF)
- SelectMenuItem(File)
  SelectMenuItem(Exit)

This scenario can be refined as:

- Select a point in the text pane by clicking in it
- Type: This is test line 1, where the selection was first made
- Type: The Enter key
- Type: The Enter key
- Type: This is test line 2
- Type: The Enter key
- Type: The Enter key
- Type: This is test line 3
- Type: The Enter key
- Select the first line of text
- Click on the Bold button
- Select the second line of text
- Click on the Italic button
- Select the third line of text
- Click on the UnderLine button
- Click on the Save RTF button
- Click the File menu in the menu bar

Click the Exit menu item

Further enhancements to the scenario:

- Start with hand where the first selection was made
- Select a point in the text pane by clicking in it
- Type: This is test line 1, where the selection was first made
- Type: The Enter key
- Type: The Enter key
- Type: This is test line 2
- Type: The Enter key
- Type: The Enter key
- Type: This is test line 3
- Type: The Enter key
- Move hand at the end of the first line of text

- Drag mouse to beginning of first line to select the entire line 1 text
- Move hand over the Bold button
- Click on the Bold button
- Move hand at the end of the second line of text
- Drag mouse to beginning of second line to select the entire line 2 text
- Move hand over the Italic button
- Click on the Italic button
- Move hand at the end of the third line of text
- Drag mouse to beginning of third line to select the entire line 3 text
- Move hand over the UnderLine button
- Click on the UnderLine button
- Move hand over the Save RTF button
- Click on the Save RTF button
- Move hand towards the File Menu Item
- Click the File menu in the menu bar
- Move hand over the Exit menu item
  Click the Exit menu item

This can be put into a sequence of independent set of actions

- Move hand from last location to new location, type: This is test line 1
- Move hand from last location to new location, type: The Enter key
- Move hand from last location to new location, type: The Enter key
- Move hand from last location to new location, type: This is test line 2
- Move hand from last location to new location, type: The Enter key
- Move hand from last location to new location, type: The Enter key
- Move hand from last location to new location, type: This is test line 3
- Move hand from last location to new location, type: The Enter key
- Move hand from last location to new location i.e. line nb1, perform dragging selection from source to destination
- Move hand from last location to new location, click on the Bold button
- Move hand from last location to new location i.e. line nb2, perform dragging action to select text from source to destination
- Move hand from last location to new location, click on the Italic button
- Move hand from last location to new location i.e. line nb3, perform dragging selection from source to destination
- Move hand from last location to new location, click on the Underline button
- Move hand from last location to new location i.e. over the Save RTF button, perform click action
- Move hand from last location to new location i.e. over the File menu, perform click action

Move hand from last location to new location i.e. over the Exit menu item, perform click action

TOISE uses this system to create the ACT-R models that are used to obtain interaction timings. It has specific primitive macros that can be launched which will create models on the fly to reflect the primitive functionality.

Table 2: TOISE 1 Interaction primitives

| **Primitive type** | **Explanations** |
|---|---|
| **(drag-drop-action-with-moveto-macro obj0X obj0Y obj0W obj0H obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)** | This is an action whereby the model will move from an earlier location to a new one where the first action of the drag action takes place. |
| **(drag-drop-Action-macro obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)** | This is an action whereby the model will perform the action of the first drag action takes place. |
| **(click-Action-with-moveto-macro obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)** | This is an action whereby the model will move from an earlier location to a new one where the first action of the click action that takes place. |
| **(click-Action-macro obj1X obj1Y obj1W obj1H thinking)** | This is an action whereby the model will perform the action of the first click action that takes place. |
| **(move-Mouse-From-To-macro obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)** | This is an action whereby the model will perform the action of moving the mouse from one place to another. |
| **(type-Action-macro obj1X obj1Y obj1W obj1H repeat-times do-a-click thinking)** | This model simulates the action of typing a number of characters at a certain location on the interface |

This is required and run those models to obtain the latencies that are necessary for the optimisation system. See Table 2.

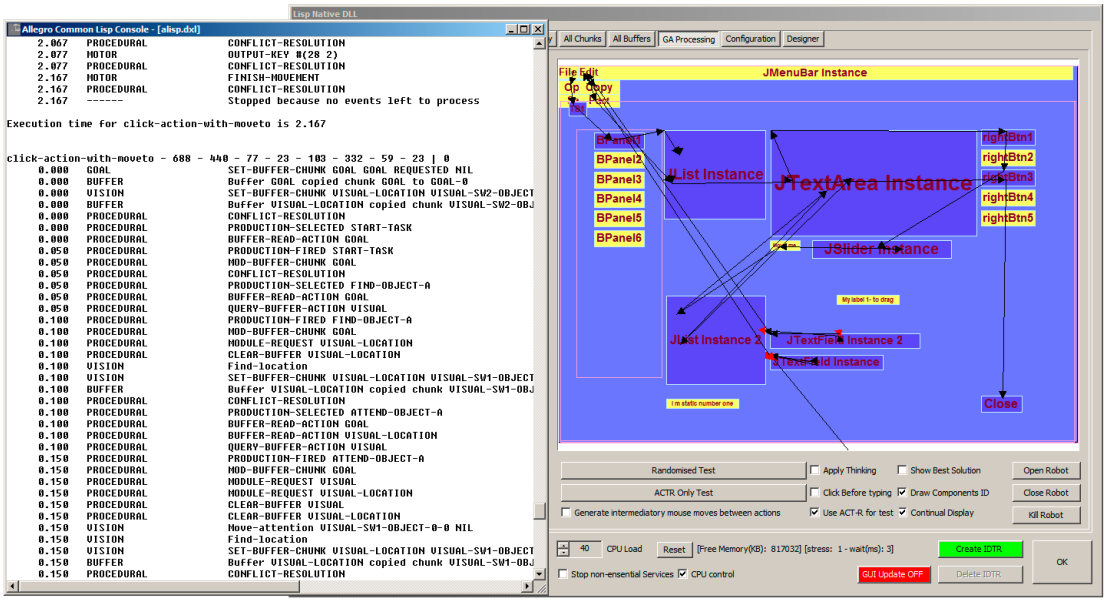



Figure 39: TOISE (Lisp/C++ version).Running model to obtain ACTR latencies

Hence a sequence of tasks can be simulated by running a sequence of these macros and thus provide the underlying functionality to allow their integration within the Genetic Algorithm fitness function which is the base of the optimisation system.

Each sub-task is sent to ACT-R for processing which results in the creation of a new model that will simulate all the necessary actions to complete this sub-task. See Figure 39.

## 5.5 System overview

TOISE development was achieved in two phases. The first prototype based on Lisp and C++ was developed to assess the underlying concepts behind ACT-R. A latter version was thereafter developed with speed in mind that contained some of ACT-R compiled into native C++ code.

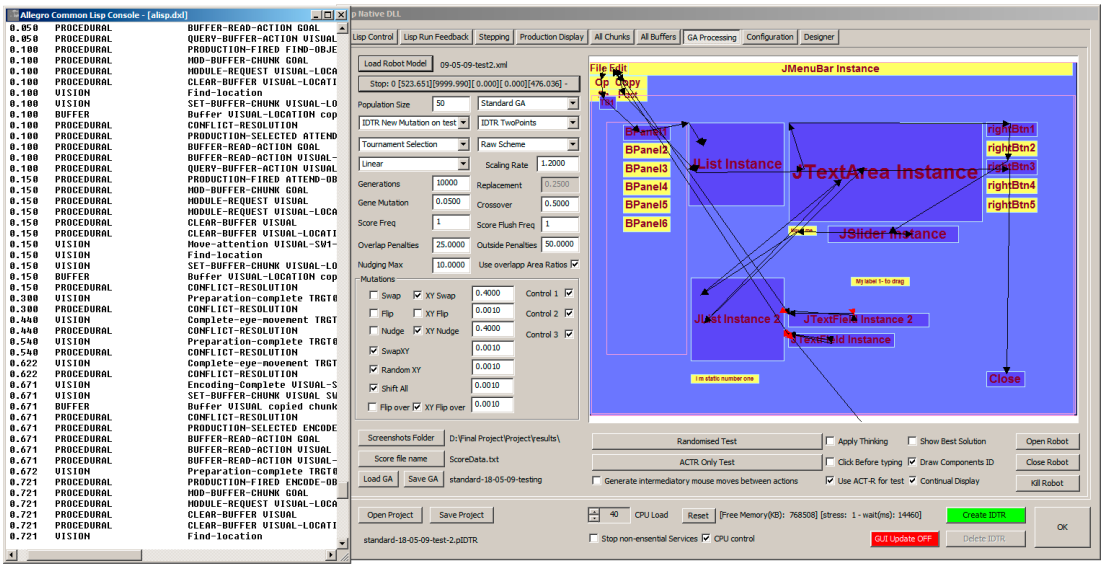



Figure 40: TOISE system

The first iteration of TOISE is displayed in Figure 40. It was based on multi-environment software that uses Lisp to access ACT-R functionality and has a C++ environment that comprised the genetic algorithm that was used as a layout optimiser. The system design is displayed in Figure 41 and its design is discussed in section 5.10. Many issues arose with this development and those are discussed in section 5.11.

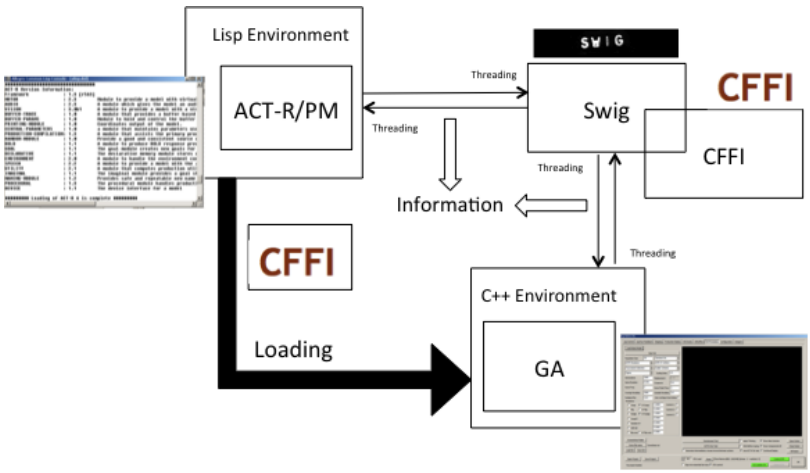



Figure 41: Lisp version of TOISE system design

## 5.6 Execution system

TOISE is more than just a Genetic Algorithm that uses ACT-R as part of its fitness function. Its first iteration was specifically created to test and evaluate the modelled interaction primitives that the system uses. The full command may include multiple sub-tasks that are separated with a "@" character. This is seen in Figure 42, which displays the running of an interaction primitive.

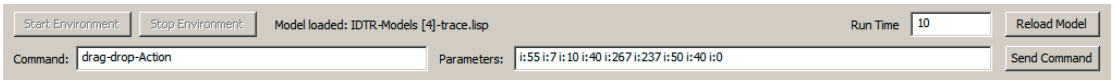



Figure 42: Running primitives

In this example, the action is to drag an object whose X & Y coordinates are 55 and 7 respectively with a width and height of 10 and 267 pixels respectively, onto an object that is located at 267 & 237 in terms of X & Y coordinates with a width and height of 50 and 40 pixels respectively. The final number i.e. 0 is a flag to allow ACT-R to add extra thinking time for this operation. The "i" tells that the data past is a number. "s" is for strings or characters. ":" is seen by the system as a delimiter.

```
(defun testfunction (data)
  (if
      (not(equal (length data) 0))
        (cmds-to_list data)
  )
)
```

Figure 43: ACT-R receiving a command from the GA

These commands are sent to the IDTR module and specifically to a function that will invoke the parsing code as seen in Figure 43.

```
(defun cmds-to_list (args)
  (let* (
    (timing)
    (new-list (string-split "@" args))
    (new-list-length (length new-list))
    (my-vector (list-to-vector-a new-list))
    (my-vector-length (length my-vector))
    (new-object-vector (make-array new-list-length) )
    )
    (setq timing 0.0)
    (dolist (item new-list)
      (let  (
      (newcmd-list (string-split "^" (coerce-string item) ))
      )
      (setq timing (+ timing (cmd-executer (car newcmd-list) (car (cdr newcmd-list)))) ) )
    )
    timing
  )
)
```

Figure 44: ACT-R parsing a series of tasks

Once the command or series of commands is received, it is parsed so it is suitable to send to an executer method that will invoke the proper macro to execute an interaction primitive. This is seen in Figure 44.

```
(defun cmd-executer (operation &rest argument-list)
    (let*  (
          (timing)
          (args (string-arguments-to_list (first argument-list)))
          (args-length (length args))
          (newOp (symbol-function  (find-symbol (string-upcase operation))))
        )
        (if (equal (first args) NIL)
          (progn
            (setq timing (apply newOp nil ) )
          )
          (progn
            (setq timing (apply newOp args )  )
          )
        )
      (format nil "~a" timing)
    )
)
```

Figure 45: Command executer for IDTR ("newop" to execute the command)

When the passed commands are formatted and properly set as a list of sequential tasks, each of the sub-tasks can be therefore executed. The method pertaining to this is shown in Figure 45.

```
(defun drag-drop-Action(obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)
    (drag-drop-action-caller obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)
    (format T "~%Execution time for drag-drop-Action is ~a ~%~%" (mp-time) )
    (mp-time)
)
```

Figure 46: Drag Drop primitive method to call specific method that invokes a macro

When the individual command are sent, the method shown in Figure 46 takes the arguments and pass them on to a specific method that invoke a macro. This design was necessary as otherwise returning the timing generated spurious errors when attempting to retrieve the operation latency.

```
(defun drag-drop-action-caller(obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)
  (setf *last-cursor-position* (vector obj1X obj1Y))
  (drag-drop-Action-macro obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)
)
```

Figure 47: Method invoking the specific macro

The method seen in Figure 47 invoke the macro that will create an ACT-R model, set the environment properly and dynamically create all the necessary chunks and goals necessary to complete the task. The Macro for this specific action is displayed in Appendix C due to its size. This macro is very similar to any ACT-R model.

- It first reset the ACT-R environment
- Command ACT-R to start with the mouse at hand
- Invoke chunk declarations
- Declares procedural rules
- List the procedural rules necessary for this primitive
- Set the ACT-R environment variables
- Dynamically create a rule if thinking is required
- Dynamically create all the visual objects that are required
- Set the ACT-R device
- Run the model
- Delete the model

Once the model is run (See Appendix D for an example), the latency is retrieved and is sent back to the calling method i.e. GA which will be used as part of the fitness assessment.

## 5.7 Debugging system

In order to ensure that the process was in line with expectations, some wrappers and hooks were implemented in ACT-R to send information back to the main GUI in the DLL loaded by Lisp for inspection as seen in Figure 48.

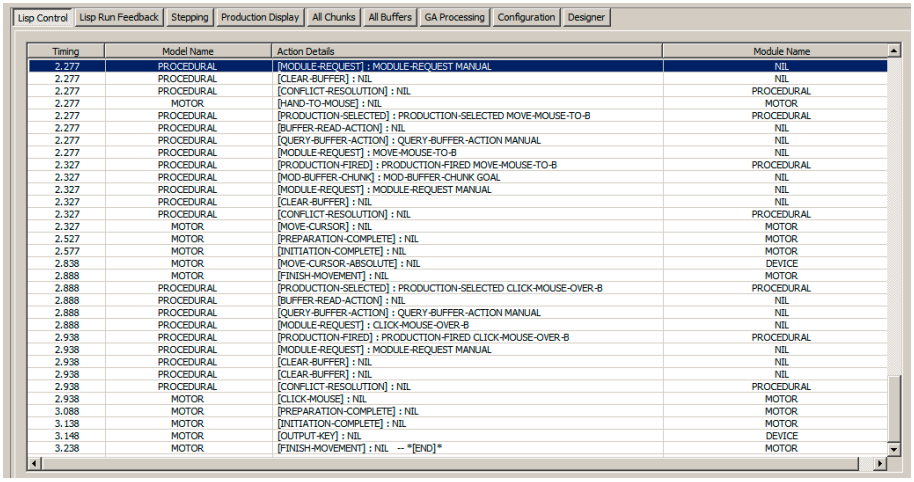



Figure 48: ACT-R Feedback windows as an ACT-R trace

Therefore when loading a sample ACT-R model representing a primitive action such as shown above, extra functionality added to the above such as described below were available:

- Stepping through the run of a model. See Figure 49 and Figure 50.
- Displaying run information whilst stepping through a model run. See Figure 51.
- Conflict resolution inner working including the *WhyNot* ACT-R functionality. See Figure 52.
- Displaying current chunks that are available in the model. See Figure 53.
- Displaying all the buffers. See Figure 54.

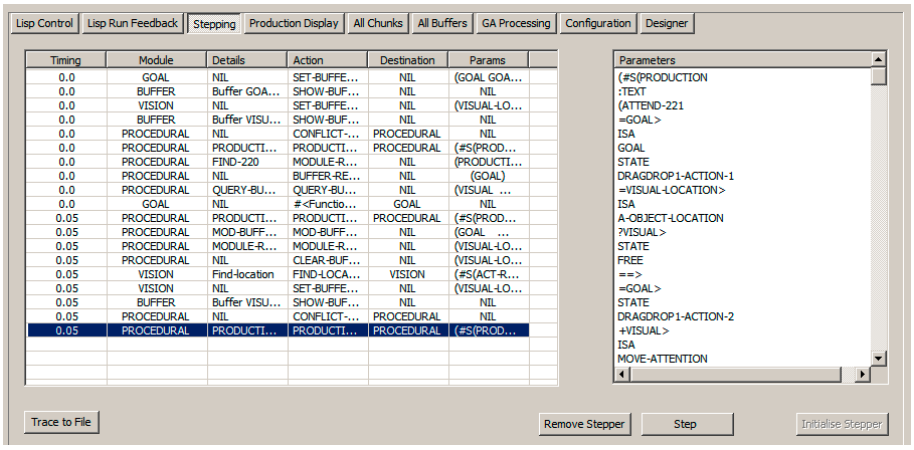



Figure 49: Stepping through a model

```
#|## Model c:\final project\project\models\dragdrop\2\drag-drop-action-2.1.lisp loaded. ##|#
     0.000   GOAL                 SET-BUFFER-CHUNK GOAL GOAL REQUESTED NIL
     0.000   BUFFER               Buffer GOAL copied chunk GOAL to GOAL-0
     0.000   VISION               SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0 REQUESTED NIL
     0.000   BUFFER               Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-0
     0.000   PROCEDURAL           CONFLICT-RESOLUTION
     0.000   PROCEDURAL           PRODUCTION-SELECTED FIND-220
     0.000   PROCEDURAL           BUFFER-READ-ACTION GOAL
     0.000   PROCEDURAL           QUERY-BUFFER-ACTION VISUAL
     0.050   PROCEDURAL           PRODUCTION-FIRED FIND-220
     0.050   PROCEDURAL           MOD-BUFFER-CHUNK GOAL
     0.050   PROCEDURAL           MODULE-REQUEST VISUAL-LOCATION
     0.050   PROCEDURAL           CLEAR-BUFFER VISUAL-LOCATION
     0.050   VISION               Find-location
     0.050   VISION               SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
     0.050   BUFFER               Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-1
     0.050   PROCEDURAL           CONFLICT-RESOLUTION
```

Figure 50: ACT-R trace whist stepping through a model run

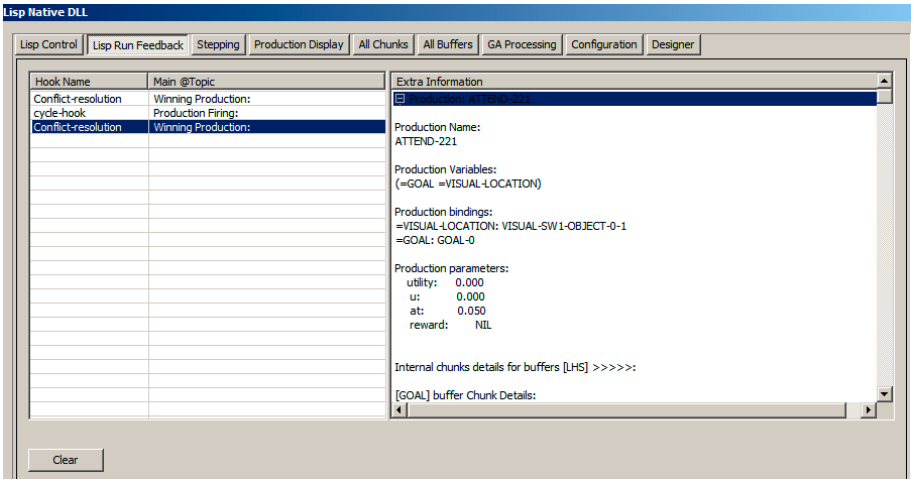



Figure 51: running information content whist stepping through a model run

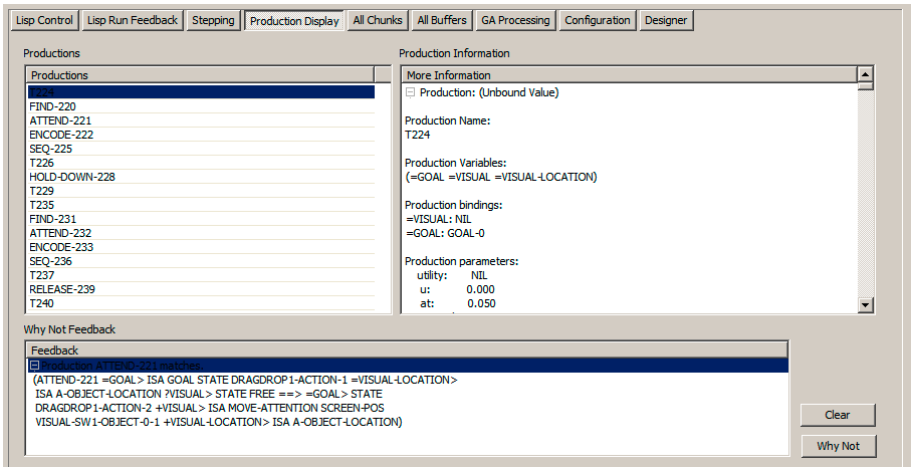



Figure 52: Procedural window contain the WhyNot functionality

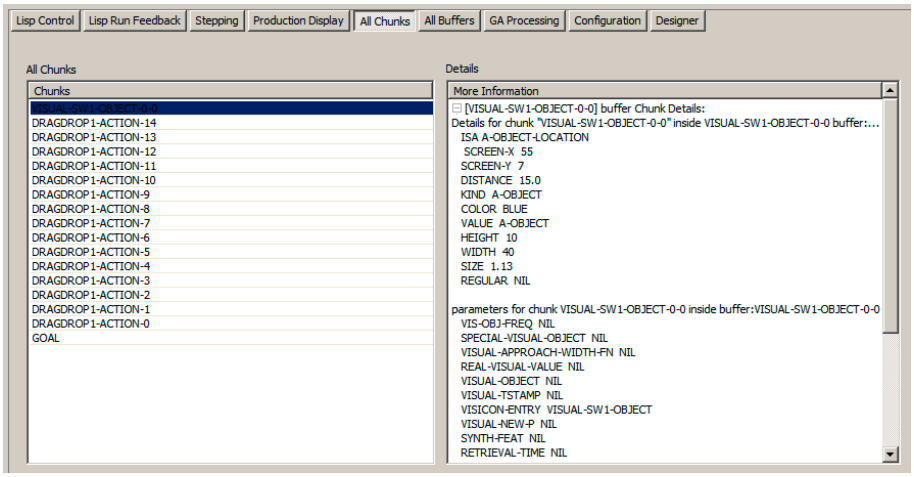



Figure 53: Chunks windows

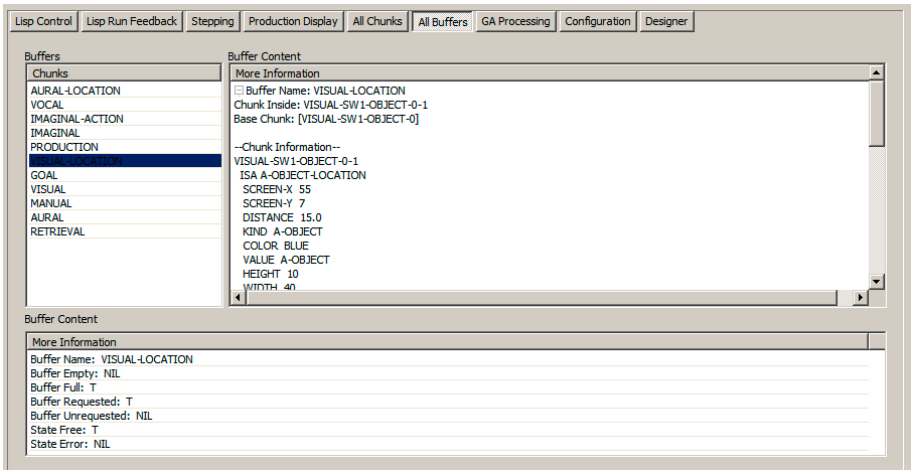



Figure 54: All Buffers windows

This extra functionality was useful to ensure the proper working of each model that is running when the optimiser i.e. the GA needed information to assess the fitness of a possible layout solution. Special methods (based on ACT-R code) have been created to

retrieve the necessary information from ACT-R and use to execute models. This code can be perused in Appendix E. In short there are seven categories as listed in Table 3.

Table 3: Categories of ACT-R coding

| Category | Explanations |
|---|---|
| **Loading ACT-R models** | The methods contained in this section deals with loading, reloading and running ACT-R models or any other suitable Lisp scripts. |
| **Stepper Code** | The code in this section allows the initialisation of a stepper system that allows stepper the running of a model. At each step, information stored in ACT-R is retrieved and displayed in specific windows of the DLL's GUI |
| **IDTR Hook helper code for information retrieval** | A special module is loaded alongside ACT-R core code to allow events and information to be retrieved from ACT-R. This is the safest and often the only way to retrieve the needed information. This functionality is used by most of the information retrieval code. |
| **Executer helper code** | This is the section of the code that manage the formatting for the information coming from the native code that deals with running models, passing the required arguments to the macros that dynamically construct and run an ACT-R model. |
| **Procedural rule information retrieval** | The related code allows the native to retrieve required information pertaining to procedural from ACT-R. |
| **Chunks information retrieval** | The related code allows the native to retrieve required information pertaining to chunks held in ACT-R. |
| **Buffers information retrieval** | The related code allows the native to retrieve required information pertaining to ACT-R buffers. |

## 5.8 Genetic Algorithm integration

Whist the first iteration of TOISE was designed to interact with ACT-R models in debugging mode or to retrieve ACT-R latencies, the main object of the native DLL was to contain the genetic algorithm that deliver the optimisation engine based on Galib (Wall 1998).

Matthew Wall from the Massachusetts Institute of Technology (MIT) created Galib [43] in the mid 90's as part of his PhD research. It is a versatile template library that includes many of the functions that are sought in a GA library. Moreover, it has a strong community and forum groups (Wall 1996; Rajanen and Jokela 2007). After some research on available genetic algorithm libraries, Galib was chosen because of its widespread use both in the industry and education institutions and its ease to manage its internal structures, which is advantageous for the type of project that we were concerned with. In addition, this library contains a strong statistical backend which is another advantage given the time constrain

[43] http://lancet.mit.edu/ga/Copyright.html

imposed on the project (Macleod 1994; Petrovic 2001; Blouin and Fadel 2002; Fahimuddin 2003; Adusumilli et al. 2007; Holzhamper and Seppelt 2007).

The inclusion of Galib into the early LISP version of TOISE native code provided the optimisation algorithm that was sought. This version was originally created to experiments with different type of crossover, mutation, selection types etc. However due to its template design, all the generic functionality was replaced by bespoke code to implement both crossover, mutation and selection operators which will be discuss in following sections when discussing the next version of TOISE i.e. named TOISE2 for short. See Figure 55.

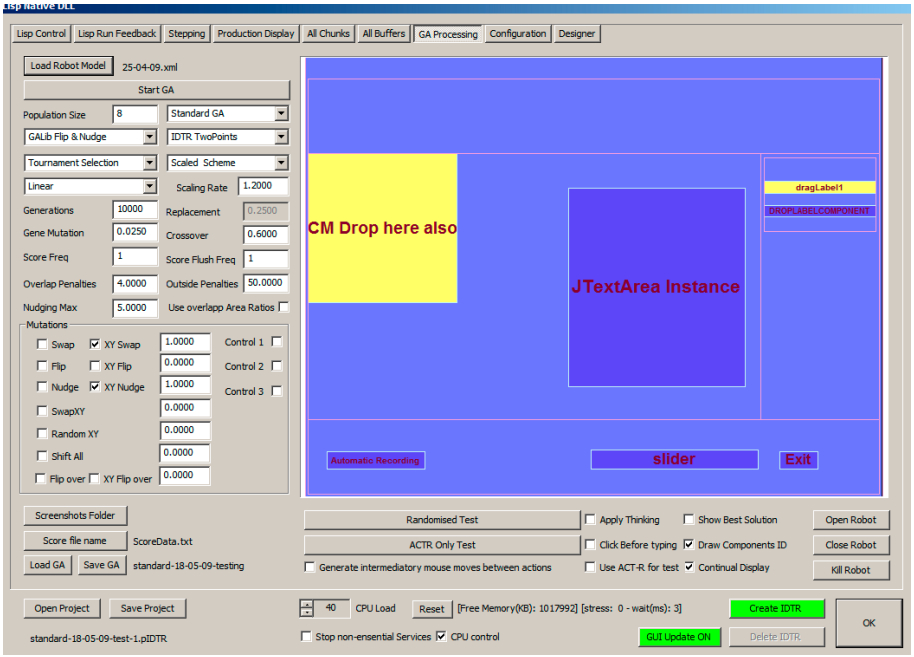



Figure 55: TOISE main GA interface prototype

## 5.9 Genetic Algorithm design

The test application that will be used to provide the necessary data for this section is shown in Figure 56. This is a simple calculator with 41 actions that were performed using simple mathematics.

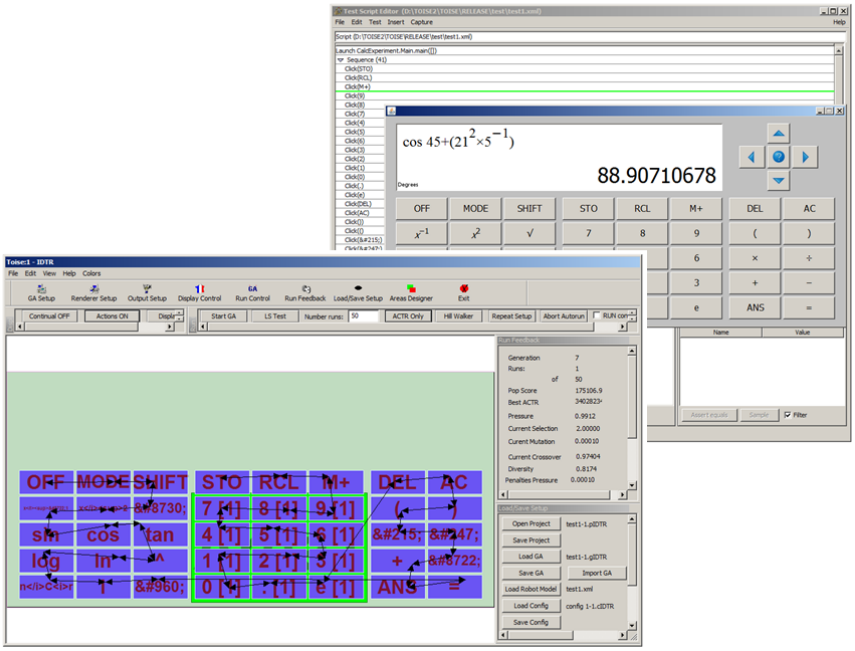



Figure 56: Test Application for testing the GA setup

In terms of interaction, the number of tasks was kept low to provide difficulties to the GA as having a small amount of interactions does not provide the pull that could render the optimisation easier to evolve. In addition, constraints were applied to the optimisation by forcing the key pad number from zero to nine, the dot and 'e' sign to remain in the centre of the area shown in green in Figure 56.

### 5.9.1 Chromosome representation

One of the main issues with using genetic algorithm is the representation of the chromosomes i.e. encoding information in a meaningful way so the GA can evolve these individuals over a number of generations.

In this project, there are two components that must be taken care of. Firstly the interface layout and secondly, the actions that are taking place on those components. The implementation this thesis focuses on is optimisation of layouts and not on optimising the user interaction as such which themselves remain constant regardless where the interface components are located, the same actions are needed. Thus the encoding should reflect the placement of the GUI components in the layout and not the user interaction itself.

The other assumption was that the metrics i.e. width and height of the components should remain constant and the only variation in terms of the layout would be their location within the layout itself.

Therefore the encoding should reflect locations of all the components; except for those that are perceived static. In TOISE, static components are layout items such as menus or invisible components such as *JLayeredPane*, which are important in terms of retrieving the layout metrics but are not interactive components that the user can interact with. However, it is possible to demote any component to a static one. The same applies to controls such as layout manager that constrain component location and sizes within an area but do not participate in the user interaction. Furthermore there are components that the user can interact with but are static and will not move; these include components such as menu or button toolbars such as JMenuBar components. The encoding of the chromosome is as follow:

$$[x_1y_1, x_2y_2, \dots, x_ny_n] \qquad (7)$$

Where $x_ny_n$ are the $x$ and $y$ coordinates of each layout's element of the interface.

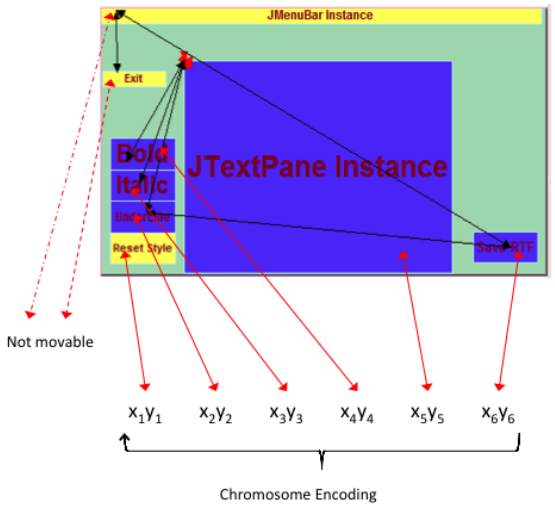



Figure 57: GA Chromosome Encoding

The encoding is explained in Figure 57. As mention above, only the layout components that can be re-located to another location are part of the chromosomes that will form the population individuals.

### 5.9.2 Genetic Algorithm generalities

In most cases, the GA parameters need to be tweaked in order to fully optimise the evolution process. With a typical GA, one may tweak the mutation rate and then tweak the crossover rate to search the search space more effectively.

Figure 58 displays how the GA evolve solutions for the given test application for different mutation and crossover rates over 250 runs of the genetic algorithm. Genetic Algorithm parameters are often solution specific but some of these parameters can also be greatly influenced by their underlying behaviour. For instance, the mutation operators are relatively speaking destructive and therefore a small mutation is expected for the GA settings. These tests shows that a mutation rate of 0.004 provides the best settings to produce solutions that have the lowest ACT-R latencies with a preference for a crossover rate of 0.7 as the distribution tends to reach the lowest value of ACT-R latencies.

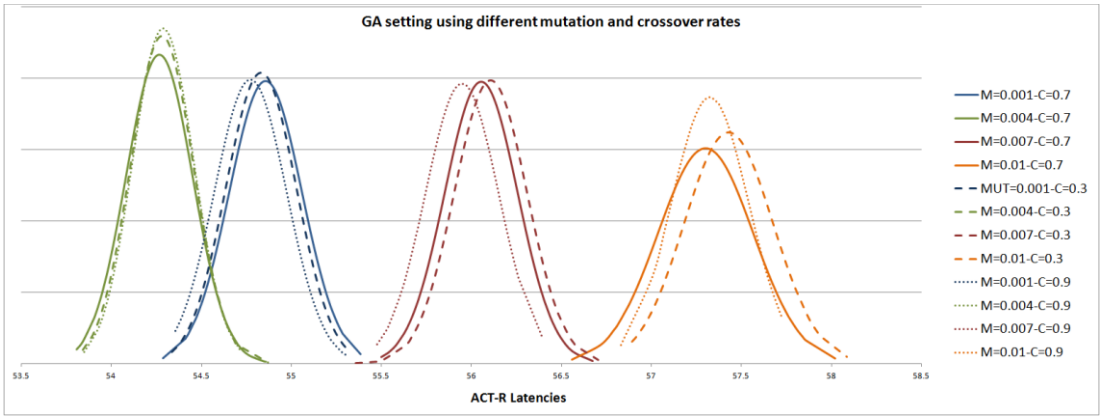



Figure 58: Tweaking the GA parameters

It is worth noting that crossover rate changes effect seems to be more pronounced at higher mutation rates than it is at lower ones. Furthermore Figure 58 seems to suggest that as the GA tuning is becoming optimal with respect to the mutation rate (and therefore the diversity of the population being kept optimised) the effectiveness of tuning the crossover rate becomes secondary in the tuning exercise.

### 5.9.3 Mending mechanism

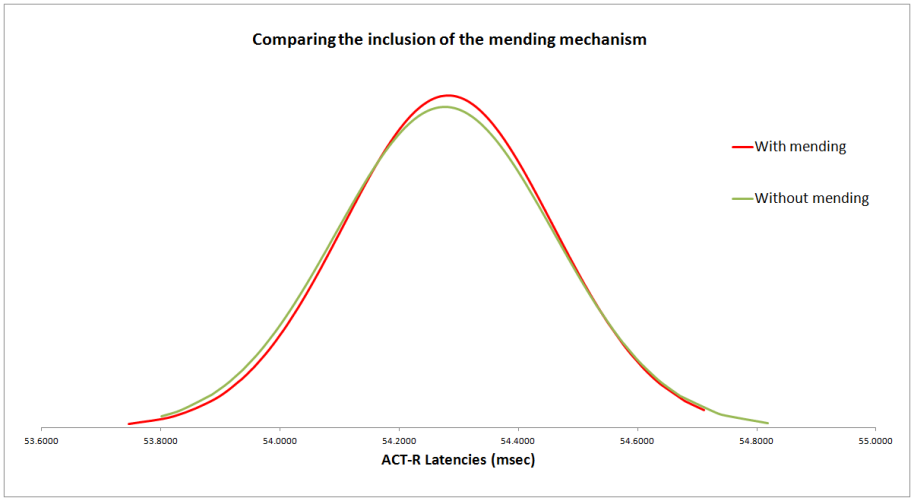



Figure 59: Relevance test of the Mending algorithm

A layout mending mechanism originally included with previous version of TOISE (Golovine et al. 2010) attempted to minimise overlapping by shifting layout components in an a manner that respected as close as possible their original location. It inherited some the mutation placement functionality discussed further in the *mutation operators* section (See Figure 65). However later version of TOISE inherited a stronger mutation mechanism and layout protection and with a more selective process, this algorithm did not prove itself necessary any longer thus freeing a considerable number of computation processes during experimentation. The original exhaustive process both in terms of experimental time and computations provided at the time to TOISE the necessary mending capability to counteract the destructive effect of the mutation and crossover operators in terms of layout feasibility. This implied though GA runs completed after days of intense computations rather than the current system that complete similar experiments in a manner of hours.

Tests were conducted (generations = 10000, mutation rate = 0.004 with base crossover rate = 0.7) to assess the benefit of the mending mechanism in TOISE. Table 4 results and t-test re-enforces the visualisation provided by the distributions displayed in Figure 59 that the mending algorithm does not lead the GA to evolve better solutions providing evidence that the inclusion of the mending system is not necessary ($t(498) = .338$, $p = .735$). It was subsequently compiled out in later versions. In conclusion, the major arguable advantage of not having this mechanism remaining in TOISE can also be expressed in terms of experimental time. The 250 experiments reported in Table 4, took 5 hours and 66 hours for the experiments not using or using layout mending respectively.

Table 4: Comparing the inclusion of the mending algorithm into the mutation process

| Individuals | Mending Mechanism | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|
| 50 | With Mending | 250 | 54.282268 | .1763215 | .0111515 |
| | Without Mending | 250 | 54.276830 | .1828835 | .0115666 |
| T-Test | | | | | |
| | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference |
| 50 | .338 | 498 | .735 | .0054376 | .0160668 |

### 5.9.4 Selection operator and dynamic operator rates

As mentioned in previous chapters, the selection system in a genetic algorithm is an important aspect of any evolutionary algorithm as it puts pressure i.e. selective pressure, on the system to single out good individuals for mating to create next generations.

Given the noisy nature of the fitness function produced by ACT-R, a *Tournament* selection was chosen for the genetic algorithm to increase this selective pressure. However there is a trade-off between using high selection pressures and maintaining a respectable level of population diversity, as high selection pressure tends to reduce the diversity of a population over time by making the population more homogenous. Conversely, lower pressure allows the diversity of a population to be maintained more easily but lowers the chances of convergence to good solutions within a reasonable amount of generations. This is compelled by the fact that genetic algorithms have a natural inherent tendency to converge rapidly, often due to low diversity as it reduces GA's ability to identify more attractive areas of the search space over a number of generations (Burke et al. 1998).

Another point to mention is that population size is also important. Higher population sizes will help to maintain the population diversity much easier than lower ones at the cost of computations. Therefore the control of population diversity which is needed is one of the most important indicator of the population state and a major factor in performance of the genetic algorithm (Lacevic et al. 2007) is associated with the amount of selective pressure that is applied to the system, the operators' rate that are used, the latter (i.e. mutation rate) often used to counteract high selective pressure by introducing new information into the individuals of an aging population and finally the population size. Figure 60 provides a visualisation of the impact of population size fluctuation with respect to GA performance. It is worth noting in these series of experiments, that increasing population size does not automatically results in significant changes in ACT-R latencies. For example, rising the GA population size from 75 to 150 individuals does not lead to any significant gains in terms of finding better quality layouts, but moving from 25 individuals to anything upwards or from 50 to 150 individuals does as shown in Table 5**.**

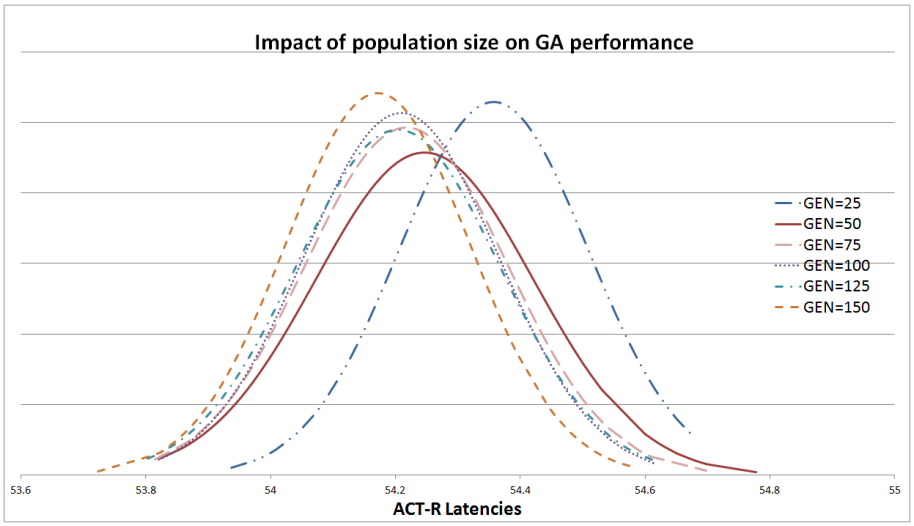



Figure 60: Impact of population size of GA Performance

To pre-empt any issues arising with regards to population diversity decreasing below a threshold that could lead the genetic algorithm to stall or to converge prematurely and therefore causing the search process to be trapped in local optima, an automatic control of the population diversity was implemented.

Table 5: ANOVA test on Population size change

| Anova Test (Turkey output) | | | | |
|---|---|---|---|---|
| **Gens** | **Gens** | **Mean Difference** | **Std. Error** | **Sig.** |
| **25** | 50 | .1104120* | .0143326 | .000 |
| | 75 | .1399273* | .0143326 | .000 |
| | 100 | .1484088* | .0143326 | .000 |
| | 125 | .1478832* | .0146421 | .000 |
| | 150 | .1854667* | .0143326 | .000 |
| **50** | 25 | -.1104120* | .0143326 | .000 |
| | 75 | .0295153 | .0143326 | .309 |
| | 100 | .0379968 | .0143326 | .086 |
| | 125 | .0374711 | .0146421 | .108 |
| | 150 | .0750546* | .0143326 | .000 |
| **75** | 25 | -.1399273* | .0143326 | .000 |
| | 50 | -.0295153 | .0143326 | .309 |
| | 100 | .0084815 | .0143326 | .992 |
| | 125 | .0079559 | .0146421 | .994 |
| | 150 | .0455394* | .0143326 | .019 |
| **100** | 25 | -.1484088* | .0143326 | .000 |
| | 50 | -.0379968 | .0143326 | .086 |
| | 75 | -.0084815 | .0143326 | .992 |
| | 125 | -.0005257 | .0146421 | 1.000 |
| | 150 | .0370578 | .0143326 | .101 |
| **125** | 25 | -.1478832* | .0146421 | .000 |
| | 50 | -.0374711 | .0146421 | .108 |
| | 75 | -.0079559 | .0146421 | .994 |
| | 100 | .0005257 | .0146421 | 1.000 |
| | 150 | .0375835 | .0146421 | .106 |
| **150** | 25 | -.1854667* | .0143326 | .000 |
| | 50 | -.0750546* | .0143326 | .000 |
| | 75 | -.0455394* | .0143326 | .019 |
| | 100 | -.0370578 | .0143326 | .101 |
| | 125 | -.0375835 | .0146421 | .106 |

Another compelling factor is that the system once developed, will be used by individuals more than likely not to be professionals into the matters of evolutionary

systems, and therefore it was felt important to remove much of the difficulties that can be encountered setting up those genetic parameters.

To pre-empt any issues arising with regards to population diversity decreasing below a threshold that could lead the genetic algorithm to stall or to converge prematurely and therefore causing the search process to be trapped in local optima, an automatic control of the population diversity was implemented. Another compelling factor is that the system once developed, will be used by individuals more than likely not to be professionals into the matters of evolutionary systems, and therefore it was felt important to remove much of the difficulties that can be encountered setting up those genetic parameters.

The algorithm is based on Zhu research (Zhu 2003) in which a dynamic genetic operators control is achieved. In this research the dynamic control is described as:

$$p' = \max(p_{min}, \min(p_{max}, p(1 + \frac{\varepsilon(d_t - d)}{d}))) \quad (5.1)$$

Where $p$ is the current rate of an operator, $p'$ is the new rate to apply to the operator at the next generation, $d$ is the diversity of the current population, $d_t$ is the target diversity, $\varepsilon$ is the control for the sensitivity for the ramping process, and $p_{min}$, $p_{max}$are the lower and upper bound of the operator's rate. $\varepsilon$ is an important parameter as it controls the rate of change in the operator rate; where a small value of $\varepsilon$ means a slow and gradual change in the rate. In TOISE, the lower and upper bounds are set to 0 and 1.0 respectively.

The adoption of the dynamic concept of diversity control in TOISE was not proposed to gain major improvements in the search capabilities of the GA compared to more traditional empirical methods of finding optimal operator rates, but its purpose was to maximise the system usability by pruning out demanding and challenging tasks both in terms of experimentation and time to set operators' rates to their optimal values. Provided very low rates exists to begin, the dynamic control mechanism, regulates those rates to ensure the population diversity remains relatively constant over a GA run thus reaching an equilibrium whist not decreasing below a specified population diversity.

To achieve this, and unlike Zhu research, TOISE goes through a first partial test run that sets those rates automatically for the following experiment runs. During this test run, the diversity drives TOISE to control all the operator rates automatically through the dynamic control, which comes into action as soon as this diversity hits a value below a specified target value. These rates are monitored and used thereafter for the main GA runs as base

rates, thus ensuring that the population diversity never goes below a level that may affect the genetic algorithm to stall and therefore decreasing its chances to converge to good solutions. The crossover rate is also allowed to adjust dynamically but it is reset every run to its default standard rate i.e. 0.7. As an example, the optimum mutation rate i.e. 0.00356 (0.004) displayed in the distribution in Figure 58 were obtained using the dynamic system, and subsequently used to set the mutation parameter for the mutation rate in a standard genetic algorithm running the same experiments.

As a conclusion, it is possible to find the optimal rates for operators using traditional approaches but the necessary systematic tests are time-consuming i.e. many tests are required with different operator rates, compared to only running the GA a quarter of the specified number of generation once with our proposed solution. Moreover, given that this prototype presents a possible tool for GUI designers that are more than likely to be remote from these types of concepts, an automated system is paramount for this concept to succeed his primary goal.

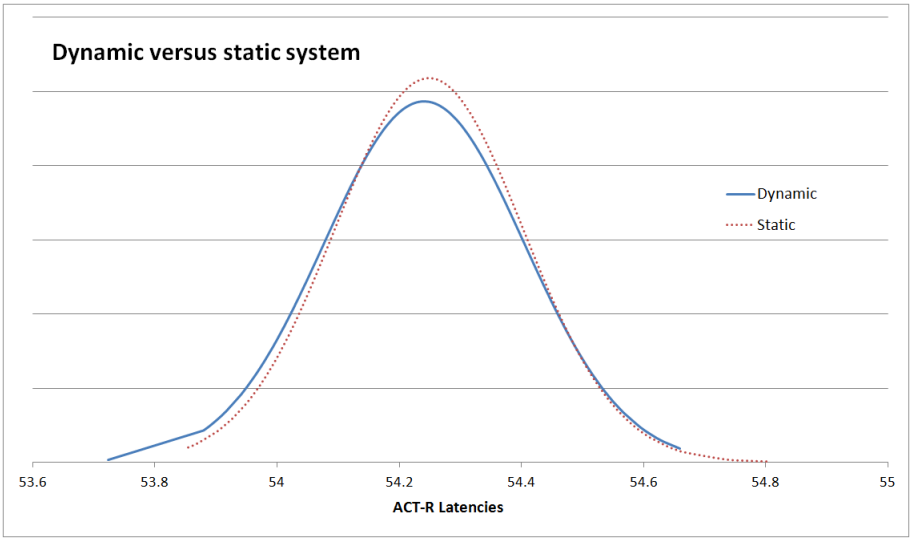



Figure 61: Dynamic operators' rates control

Experiments were carried out to assess this mechanism compared to a classic GA. These experiments consisted of running an experiment with a mutation rate that is relatively low i.e. 0.0001 for the dynamic GA and a known pre-optimised mutation operator rate i.e. 0.004 for the classical GA; the hypothesis being that no significant mean difference exists between the two systems. The crossover was set to 0.7 for both systems. Results are shown in Figure 61 and Table 6. Looking at Figure 61 the run distribution seems to suggest that the results re-enforce the assumption that using a dynamic system does not impair the genetic algorithm in its capacity to find good layouts but it does increases its chances by virtue of setting the operator rates optimally.

| Table 6: Dynamic versus static mutation control | | | | | |
|---|---|---|---|---|---|
| **Individuals** | **Algorithm** | **N** | **Mean** | **Std. Deviation** | **Std. Error Mean** |
| **50** | **Dynamic Mutation** | 249 | 54.256041 | .1754756 | .0111203 |
| | **Static Mutation** | 249 | 54.272237 | .1749176 | .0110850 |
| **T-Test** | | | | | |
| | **t** | **df** | **Sig. (2-tailed)** | **Mean Difference** | **Std. Error Difference** |
| **50** | -1.031 | 496 | .303 | -.0161960 | .0157015 |

Tests were also conducted to assess if the presence of a test pre-run affects the ability of the genetic algorithm negatively, as this is a departure from a standard GA, with the assumption that it did not (generations = 10000, mutation rate = 0.00001 with base crossover rate = 0.7). In addition, there were many strategies that were available.

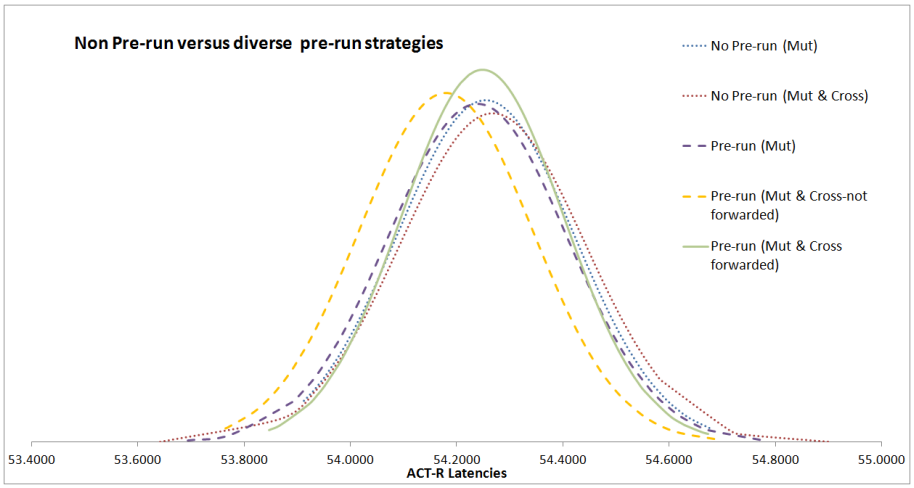



Figure 62 : No pre-run versus pre-run strategies

In Figure 62, the distribution curves provides evidence that using a pre-run with both mutation and crossover rates controlled by the dynamic system provides the best solution if the fluctuated crossover rate is not used for subsequent runs as a base rate but is restored to its default value at each start of the runs. All but one strategy seems to suggest no significant changes in terms of ACT-R latencies. Table 7 results re-enforce this hypothesis ($t(475) = -4.760$, $p = .000$).

| Table 7: Pre-run strategies (with or without carrying crossover rate to other runs) | | | | |
|---|---|---|---|---|
| **Algorithm (pre-run)** | **N** | **Mean** | **Std. Deviation** | **Std. Error Mean** |
| **Mut & Cross-not forwarded** | 242 | 54.179851 | .1631685 | .0104889 |
| **Mut & Cross forwarded** | 235 | 54.248797 | .1528197 | .0099689 |
| **T-Test** | | | | |
| **t** | **df** | **Sig. (2-tailed)** | **Mean Difference** | **Std. Error Difference** |
| -4.760 | 475 | .000 | -.0689466 | .0144844 |

There is however another aspect to take into account with controlling the crossover rate dynamically. As the automatic mechanism adjusts the operators' pressure depending on the diversity of the population, these fluctuations either increase or decrease the number of computations that the system has to achieve. Experiments were conducted to verify the assumption (generations=10000, base mutation rate = 0.004, base crossover rate = 0.7) that when dynamically controlled, the number of crossover evaluations is decreased substantially thus releasing computer resources as well as benefiting from shorter run times.

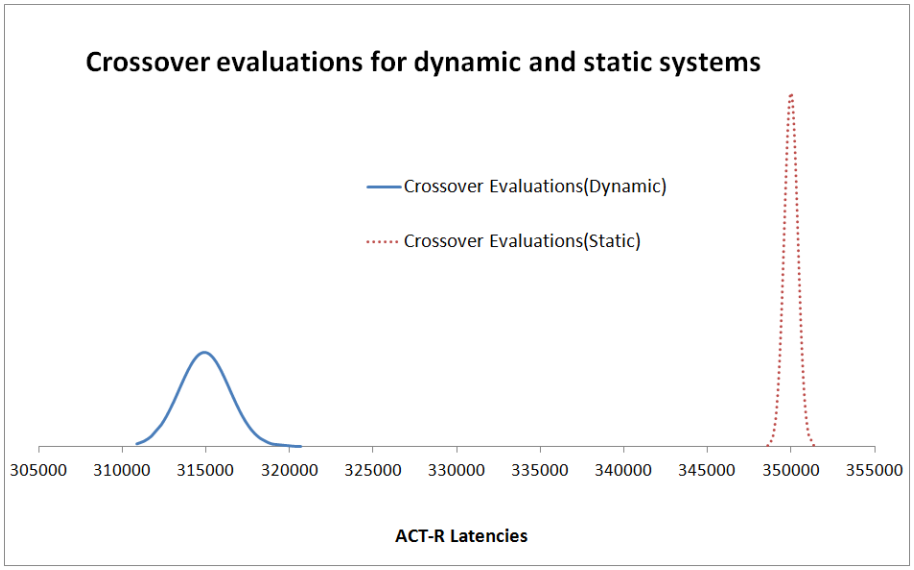



Figure 63: Dynamic control - Crossover evaluations

Figure 63 confirms that indeed the number of crossover evaluations are significantly lower when the crossover rate is dynamically controlled; in this example by around 11%. Those results were confirmed by a t-test shown in Table 8 ($t(234.942) = 289.200$, $p = .000$). This adds to the point already mentioned, that the addition of controlled crossover rate adds advantages to the system.

Table 8: Dynamic control for testing crossover evaluations

| Algorithm | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|
| Dynamic Crossover Evals | 249 | 315099.493976 | 1714.8906156 | 108.6767770 |
| Static Crossover evals | 250 | 350053.040000 | 450.9339955 | 28.5195700 |
| | | T-Test | | |
| t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference |
| -311.095 | 282.002 | .000 | -3.4953546E+04 | 112.3566097 |

The final aspect of dynamic operator control is the hypothesis that larger values of *sensitivity* favour the hopping from one local optimum to another by the search process hence increasing the chances to find better solutions as mentioned in Zhu's body of work.

In simple terms, on one hand the system reacts smoothly to changing population diversity but and on the other pronounced shifts of operators' rate are applied.

This hypothesis was tested by a series of 250 experiments ranging from the suggested values of 0.1 down to 0.01 i.e. 0.01, 0.003, 0.006 and 0.1, for the diversity control sensitivity in Zhu paper (generations=10000, base mutation rate = 0.0001, base crossover rate = 0.7). Figure 64 suggests that changing the rate of change of the sensitivity has an impact on the overall GA capability to search for good layout with both 0.06 and 0.03 being the best and worth respectively for this series of experiments whilst the high and low bounds suggested by Zhu are not in this case the optimum values but provide a middle of the road values that are problem specific. However , Table 9 and Table 10 confirm that setting the optimum value for this setting does provide a difference of means from 54.261146 seconds down to 54.179147 seconds; a difference of around 82ms which in terms of human cognition is a big number; (t (271) = .4.078, p = .000). However to find this optimum rate with the proposed dynamic system, is much easier and quicker to achieve for a novice user.

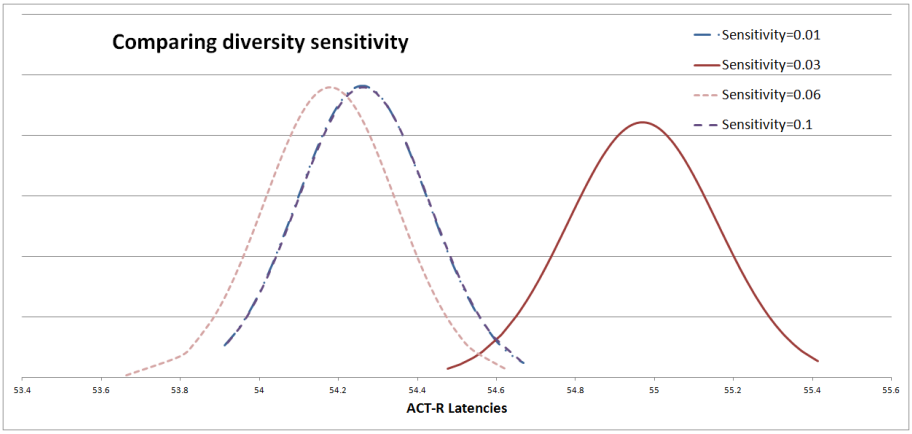



Figure 64: Sensitivity for operators' control set from 0.1 down to 0.01

Table 9: Sensitivity for operator control [0.1,0.01]

| Anova Test (Turkey output) | | | | |
|---|---|---|---|---|
| **Sensitivity** | **Sensitivity** | **Mean Difference** | **Std. Error** | **Sig.** |
| **0.01** | 0.03 | -.7088443* | .0190285 | .000 |
| | 0.06 | .0819987* | .0212012 | .001 |
| | 0.1 | -.0023102 | .0219033 | 1.000 |
| **0.03** | 0.01 | .7088443* | .0190285 | .000 |
| | 0.06 | .7908430* | .0187232 | .000 |
| | 0.1 | .7065340* | .0195147 | .000 |
| **0.06** | 0.01 | -.0819987* | .0212012 | .001 |
| | 0.03 | -.7908430* | .0187232 | .000 |
| | 0.1 | -.0843090* | .0216386 | .001 |
| **0.1** | 0.01 | .0023102 | .0219033 | 1.000 |
| | 0.03 | -.7065340* | .0195147 | .000 |
| | 0.06 | .0843090* | .0216386 | .001 |

| Table 10: Comparing sensitivity = 0.01 and 0.06. | | | | |
|---|---|---|---|---|
| **Algorithm** | **N** | **Mean** | **Std. Deviation** | **Std. Error Mean** |
| **Sensitivity=0.01** | 133 | 54.261146 | .1655980 | .0143592 |
| **Sensitivity=0.06** | 140 | 54.179147 | .1664706 | .0140693 |
| **T-Test** | | | | |
| **t** | **df** | **Sig. (2-tailed)** | **Mean Difference** | **Std. Error Difference** |
| 4.078 | 271 | .000 | .0819987 | .0201057 |

### 5.9.5 Mutation operators

The use of Galib allows for the development of bespoke operators. In many GA solutions, these operators operate on individual genes. However in TOISE, these operators operate on sequential pairs of genes. The reasons behind this choice was that operators can disturb the chromosome in harmful way i.e. be too disruptive. Thus the choice was made to mutate genes as XY pairs thus respecting their pairing as they refer to their X and Y coordinates whilst minimising disturbances.

The mutation system is entirely based on random choices. First, a selection of pairs of genes is randomly made which represents components that can be nudged on the layout. Second, a mutation operator is selected at random; an operator that will mutate the selected genes in a manner specified by the type of mutation the operator pertains to.

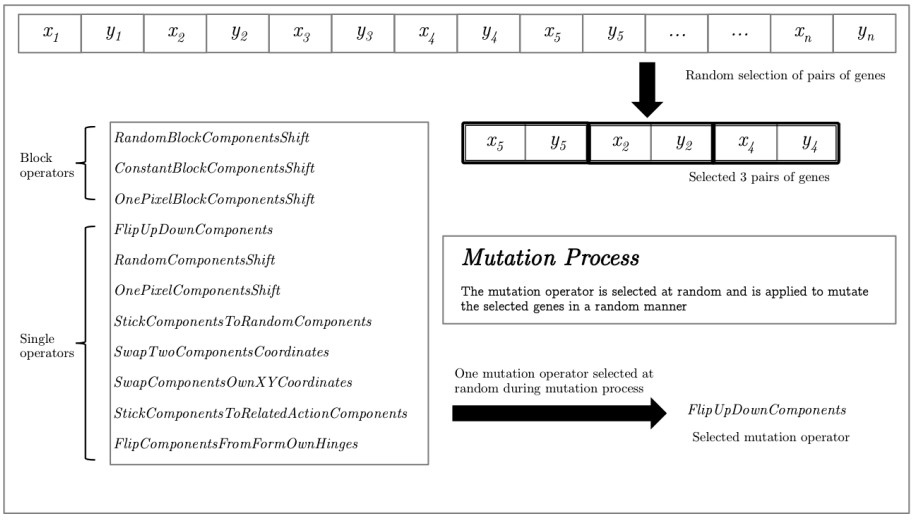
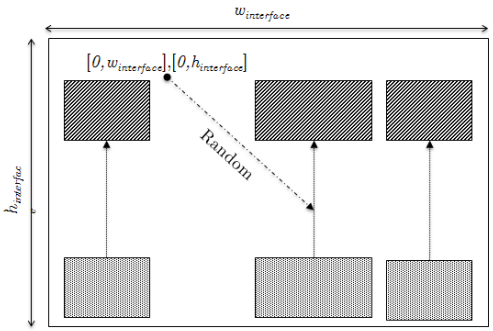



Figure 65: TOISE Mutation system

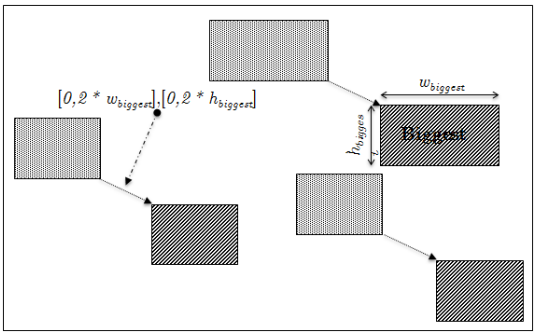
| Table 11: The 12 Mutation Operators | |
|---|---|
| RandomBlockComponentsShift | Operator shifts all selected genes in the same direction by the same random amount that is randomly chosen to be within the interface area. See Figure 66(a). |
| ConstantBlockComponentsShift | Operator shifts selected genes in the same direction by the same constant amount, which equals to double the size of the biggest sized component. See Figure 66(b). |
| OnePixelBlockComponentsShift | Operator shift selected genes in the same direction by the same constant amount set to 1 pixel. See Figure 66(c). |
| FlipUpDownComponents | Operator select a pair of X and Y genes and flip it from top to bottom, left to right etc. See Figure 66(d). |
| ConstantComponentsShift | Operator shift selected pairs of genes in the different direction by a constant amount, which equals the size of the biggest sized component. See Figure 66(e). |
| RandomComponentsShift | Operator shifts all selected genes in the any random direction by an individually computed random amount that is randomly chosen to be within the interface area. See Figure 66(f). |
| OnePixelComponentsShift | Operator shift selected genes in any random direction by an individually computed amount set to 1 pixel. See Figure 66(g). |
| StickComponentsToRandomComponents | Operator selects a pair of X and Y genes at random, then selects another pair in the same manner, and sticks the former component close to the former in a random direction taking into account their metrics. See Figure 66(h). |
| SwapTwoComponentsCoordinates | Operator selects at random two pairs of gene and swaps their X and Y coordinates values. See Figure 66(i). |
| SwapComponentsOwnXYCoordinates | Operator selects a pair of sequential genes referring to X and Y coordinates of a component and swaps those coordinates. See Figure 66(j). |
| StickComponentsToRelatedActionComponents | Operator selects a pair of genes at random which relates to a layout component, then follows the trail of actions performed on this component to find the component that is related with an action and will attempt to stick closer to its location X and Y coordinated in a random direction. See Figure 66(k). |
| FlipComponentsFromFormOwnHinges | Operator selects a hinge from a component and flips it using width or height depending of the direction which is determined in a random manner. See Figure 66(l). |

TOISE provides two overall types of mutation operators. Some operators operate on individual pairs of genes that can be shifted in any direction i.e. left, right, top, bottom, or diagonally by some random distance, thus moving some components on the layout in an individual manner whilst the block operators still operate on a selection of pairs of genes but shift some selected components on that layout in the same direction by the same amount.

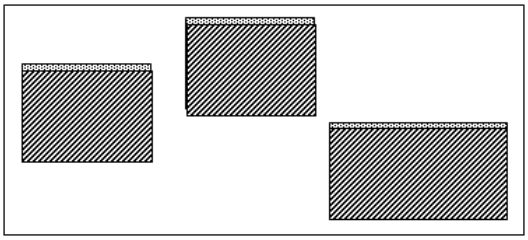
Their functionality is explained in Table 11 and a diagram of the mutation system is presented in Figure 65. An explanation of the way the mutation operators operation is explained in Figure 66.



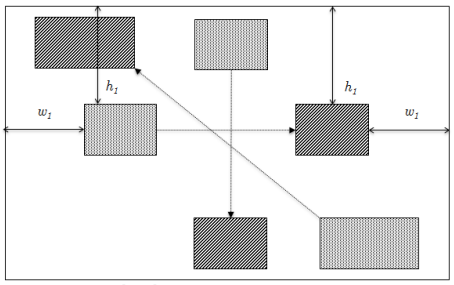

( a ) RandomBlockComponentsShift



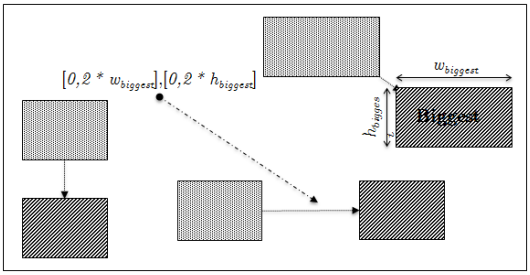

( b ) ConstantBlockComponentsShift



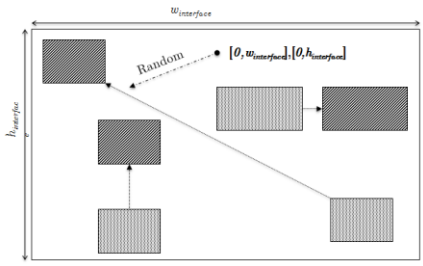
( c ) OnePixelBlockComponentsShift



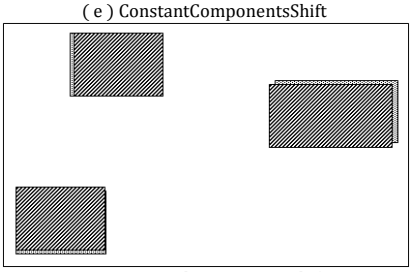

( d ) FlipUpDownComponents



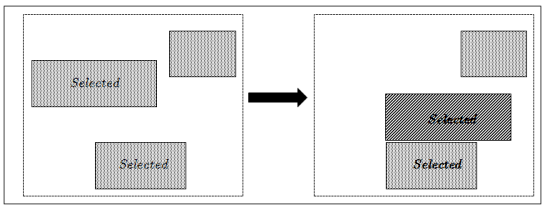

( e ) ConstantComponentsShift



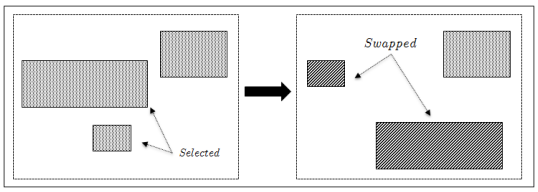

( f ) RandomComponentsShift



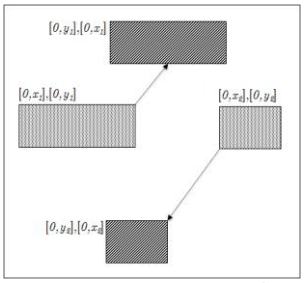
( g ) OnePixelComponentsShift



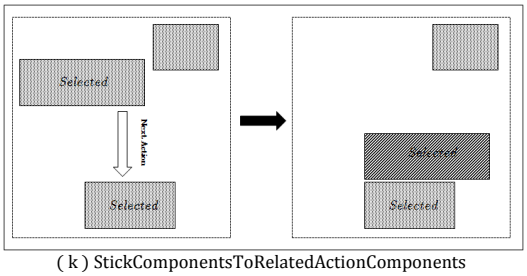

( h ) StickComponentsToRandomComponents



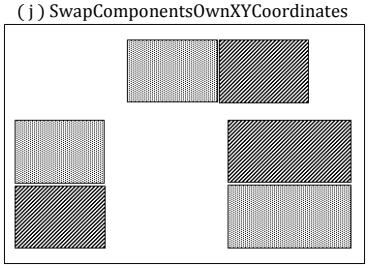

( i ) SwapTwoComponentsCoordinates




( j ) SwapComponentsOwnXYCoordinates




( k ) StickComponentsToRelatedActionComponents



( l ) FlipComponentsFromFormOwnHinges

Figure 66 Mutation operator operations

TOISE like any GA system relies on its mutation algorithm(s) to provide the exploration of the search space and finding new layout configurations over a number of generations. These small and subtle changes in the genes' configuration provide the mechanism that brings into an existing population necessary changes to gradually evolve the layout in the manner that minimise the cognitive and motor loads when a set of actions is apply to these components.

To evolve this layout, the layout components must be able to find a location within an amount of constraints that makes the layout workable. Therefore these components must remain within the general area of the layout, not overlap with other components and if constrained to be within an area, they must be located and optimised within the specified area. It must be also noted that some components should remain static i.e. not movable. Components such as menu bars and their menu items are located to the top of the application; a template that is followed across operating system platforms. Hence the mutation system must also take into account these constraints too.

The mutation operators from which one is selected at random, delivers the tool that allows a component on the layout to be nudged to another location. The mutation system only ensures that if a component is shifted, it must remain with the general layout area. If not it is nudged to the closest location that ensure it is within this area in a fashion that takes into account the random direction a component was supposed to be nudged. During this process, there are no attempts to check if overlapping or braking area rules occurs as this is one of the assessments that is taking care off during fitness assessments where penalties are imposed if instances such as this occur ultimately impacting on the selection of that individual during the selection rounds.

Perusing at Table 11 from figures (a) to (l), one can see that the operators were conceived to help a component to shift to any place in the layout or to find a place that is relevant. For instance the *random, swap* and *flip* operators are the more exploratory operators, whereas the *nudging* and the *stick* operators are more the consolidator operators that make sure those components that are related by an action or areas move closer to each other.

The mutation system is described in details in Appendix F. Overall there are a few steps:

- Retrieve the genes from the main GA.
- Select a random pool of genes that will undergo mutation
- Select a mutation operator at random
- Invoke the mutation operator to operate on selected pair of genes
- Retrieve and pass to Galib the total number of mutations that took place

The entire process is based on random selections but an exception is made regarding the selection of the genes, which are selected by pairs to reflect the X, and Y coordinates of components. The main coding in terms of methodology is based on Galib but the code was amended to reflect the operators' functionality in the real world i.e. component of a layout. Thus instead of mutating individual genes individually, a pair of sequential genes are taken and mutated together by the same process. This minimises also the disturbance effects of the mutation process yet moving component on the layout in a sensible way i.e. when one does move or place a component on a layout, one affects both the X and Y coordinates during this action. Since we are simulating human cognitive and motor functions during the assessment of a layout, it makes sense to place the widgets in a similar manner. See Figure 67.

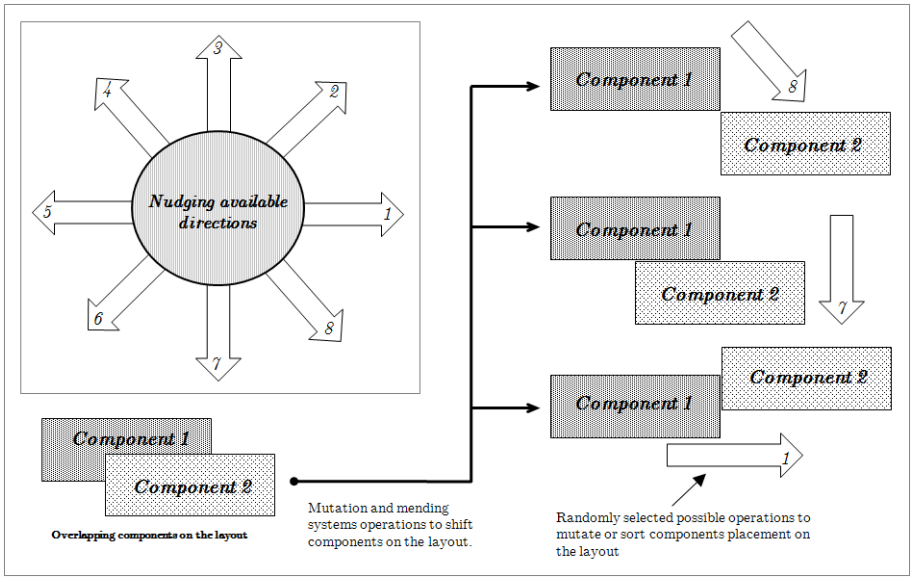



Figure 67: Mutation system and control

Some care was taken to ensure that a component could move in any direction. A set of directions is randomly chosen i.e. whether the component should be moved up or down, left or right or combining both to move diagonally. All mutation operators use this system to nudge components in their new locations. An example of such operator is displayed in Appendix G.

The mutation operator selection changes strategy, which is based on the concept that mutation operators are chosen at random over the range of available operators and therefore all the operators have the same probability of selection. However, as a feasible layout is evolved, i.e. no components are located outside the main interface area, all components are sitting in their own area if required, and little overlapping is showing, a change of strategy occurs to drive the evolution process to consolidate a layout configuration when low interaction pressure is applied to the evolution system. In this work, this is called mutation operator selection biasing.

The mentioned strategy is to provide, once a feasible layout is found, the *sticking* operators a little more probability to be selected in the mutation operator selection process so some consolidation of the layout can occur (Given that ACT-R does not calculate the Fitt's law if a change in visual angle is smaller than five degrees). This allows some related objects in a layout to come closer to each other thus minimising in a meaningful way human motor processes (refer to the Fitt's Law section) and as a consequence the ACT-R latencies. (The mutation process code is displayed in Appendix F, and the code highlighted in green refers to the strategy discussed in this section.). Figure 68 describes the process.

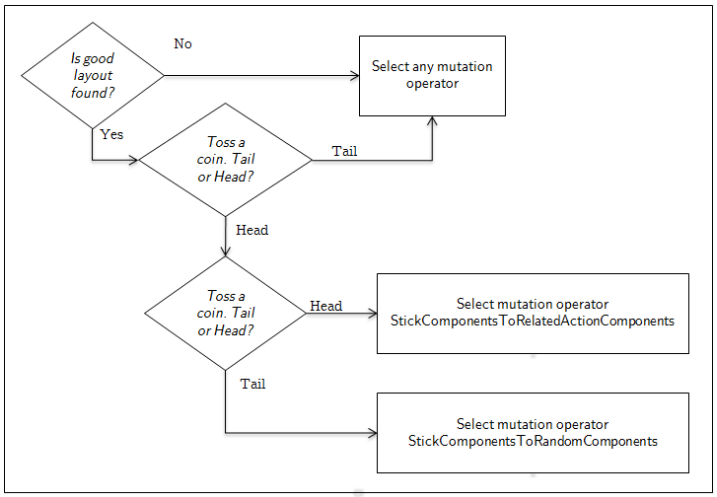



Figure 68: Mutation biasing algorithm

### 5.9.6 Crossover operators

The crossover operators are based on Galib and were developed in the same manner as the mutation operators. Because of the need of consistency with the mutation system the crossover operators transfer information from pairs of genes from one generation to another thus respecting the pairing of X and Y coordinates for the components of layout. An example of the code for a 2-points crossover is shown in Appendix H. The main types of crossovers such as *EvenOdd*, *Uniform*, *1-point* and *2-points* are available but the standard chosen is the *2 points* crossover type.

### 5.9.7 Fitness Assessment

The fitness function is invokes each time an individual of a genetic algorithm is entering the selection competition. Thus it is a crucial part of a GA. Because there are many aspects that TOISE needs to assess before returning an individual's fitness, the overall strategy for the fitness function is based on multi-objective fitness, which has been described in the literature. Genetic algorithms are well suited for multi-objective optimisation both because of their parallel search mechanism; where multiple individuals can search the solution space for multiple solutions in parallel, taking advantage of any similarity amongst possible solutions, and their effectiveness at handling complex problems with noisy evaluations and discontinuous search space (Fonseca & Fleming 1995).

TOISE uses a known used technique, which is the weighted aggregating approach; an approach that is straightforward to implement and is computationally effective (Konak et al. 2006), where summing up the weighted objective values that are part of the overall fitness assesses the fitness of an individual. With this technique, maintaining diversity within the population is crucial as the genetic algorithm deals with more isolated optima given the many objectives that are part of the final assessment and the possibility of dominance of some of the objectives over others over some part of the run time which often lead to local optima (Zitzler et al. 2000). In a general form this type of optimisation problem can formulated as:

$$Y = f(x) = (f_1(x), f_2(x), f_3(x), \dots f_k(x))$$
$$\text{given} \quad x = (x_1, x_2, \dots x_m) \in X \quad \& \quad y = (y_1, y_2, \dots y_n) \in Y \tag{5.2}$$

Where $f$ is a vector function that maps a set of $m$ parameters (decision variables) to a set of $n$ objectives. $x$ is a set of the decision vectors and $X$ is the parameter space, $y$ is the objective vector, and $Y$ is the objective space. Therefore, the set of solutions of a

multiobjective optimisation problem consists of finding all the decision parameters for which corresponding objective vectors cannot be refined in any dimensions without degradation in another. These types of optimisation produce a set of unique equally good solutions known as pareto-optimal solutions (Zitzler et al. 2000).

In the context of TOISE, the parameters are a set of component locations in a layout to optimise i.e. $z = (x_1, y_1, x_2, y_2, \dots, x_n, y_n)$ such that:

$$f(z) = w_a C(z) + w_b O(z) + w_b A(z) + w_b W(z) \quad (5.3)$$

Where $C(z)$ is the task latency provided by ACT-R, $O(z)$ is a penalty given for overlapping components that is proportional to the area of overlap, $A(z)$ is a fixed penalty proportional to the amount of components that are located outside the layout boundaries, and $W(z)$ is a fixed penalty proportional to the amount of area that is outside dedicated areas (if requested). Each objective has a weight coefficient $w_i$that is controlled through the interface. In TOISE, $w_a$ is set to 1.0 per default and $w_b$ is set to 5000 as default value.

As with many real-life problems, individual objectives conflict with each other and TOISE is no exception. For instance, optimising $z$ with respect to a single objective such as $C(z)$ will often results in unsatisfactory results with respect to other objectives such as $O(z)$, $A(z)$,$W(z)$ i.e. constraining components within an area of the layout often lead to inflated ACT-R motor and cognitive latencies and therefore reduces the search space. Thus satisfying all the objectives may lead to compromises in terms of layout solution quality.

#### 5.9.7.1 Overlapping penalty mechanism

The overlapping mechanism is based on the component overlapping area that is accumulated and returned as a fraction compared to the total area that is occupied by all the components. This is a simple algorithm but very effective at picking up component overlap. The code can be perused in Appendix I. The overall method is as below:

- For each component in the layout that can be moved
- Increase total area by the area of the component
- Calculate the overlap with other components
- Increase the total overlap area if overlapping
- Finally returns the fraction of overlap compared to total overlap

#### 5.9.7.2 Outside the layout area penalty mechanism

The algorithm for this penalty is very similar to the overlapping penalty system with the difference that instead of accumulating the area that is only outside the area, the aggregation adds the entire area of the component. This difference means that more pressure will incur when a component is outside or partially outside the layout as it would if only the out of bound area was only added. The code can be perused in Appendix I.

- For each component in the layout that can be moved
- Increase total area by the area of the component
- Increase the total out of bounds area by component are if outside the layout
- Finally returns the fraction of out of bounds area compared to total overlap

#### 5.9.7.3 Outside the layout designated area penalty mechanism

In TOISE, any area can be created on the layout and component can be assigned one of those areas for which it will bond to. If one of these components is located outside the designated area, a penalty occurs. Unlike the other two penalty mechanisms, this algorithm does not deal with proportional values. It returns accumulated fixed penalties for breaching area rules. A fix penalty of 6 points is given for breaching area boundaries. However points are taken off given some criteria that are discussed in Table 12, whilst Figure 69 displays those concepts.

In addition to its own area, penalties are applied if a component is placed in an area that it is not bound to with a penalties of six points. This helps the GA to shift the components in their right areas much quicker and avoid situations where a component is trapped into an area which should be occupied by another component. This is needed as otherwise, there is no gain for the GA in terms of fitness to move this component because it is not breaching *out of areas* rules and the cost of overalapping if moved somewhere else is much cheaper than remaining in this un-authorised location.

| Table 12: Out of area penalty system | |
|---|---|
| **Penalties (+ or -)** | **Explanation** |
| **Full penalty = 6** | A component that partially or completely lies outside its designated area is given 6 point penalties. See Figure 69(a) |
| **Penalty =- 1** | A component whose center lies within the designated area i.e. half of the component is within its area will see the penalty decreased by 1. See Figure 69(b). |
| **Penalty =- 1** | A component that has its bottom right corner within its area sees the amount of penalty points decreased by 1. See Figure 69(c) |
| **Penalty =- 1** | A component that has its bottom left corner within its area sees the amount of penalty points decreased by 1. See Figure 69(d) |
| **Penalty =- 1** | A component that has its top right corner within its area sees the amount of penalty points decreased by 1. See Figure 69(e) |
| **Penalty =- 1** | A component that has its top left corner within its area sees the amount of penalty points decreased by 1. See Figure 69(f) |

### 5.9.8 Conclusion

The fitness function behaviour will return high values as long as the layout components lay outside the main area of the interface, or when components lie outside their designated area. These two components of the fitness function are the drivers, compared to overlapping, due to their high cost at the beginning of the evolutionary process which very quickly through selection lead the genetic algorithm to evolve solutions i.e. layouts, that exclude component arrangements that include components outside the interface or specified areas.

As the GA run goes along, the overlapping penalties which were of minor importance to begin with compared to other penalties, begin to influence the evolutionary system and become more substantial and in par with ACT-R Latencies, therefore leading the genetic algorithm selection algorithm to select layouts that do not contain component overlapping with others but also layouts that compete in terms of cognitive and motor load.

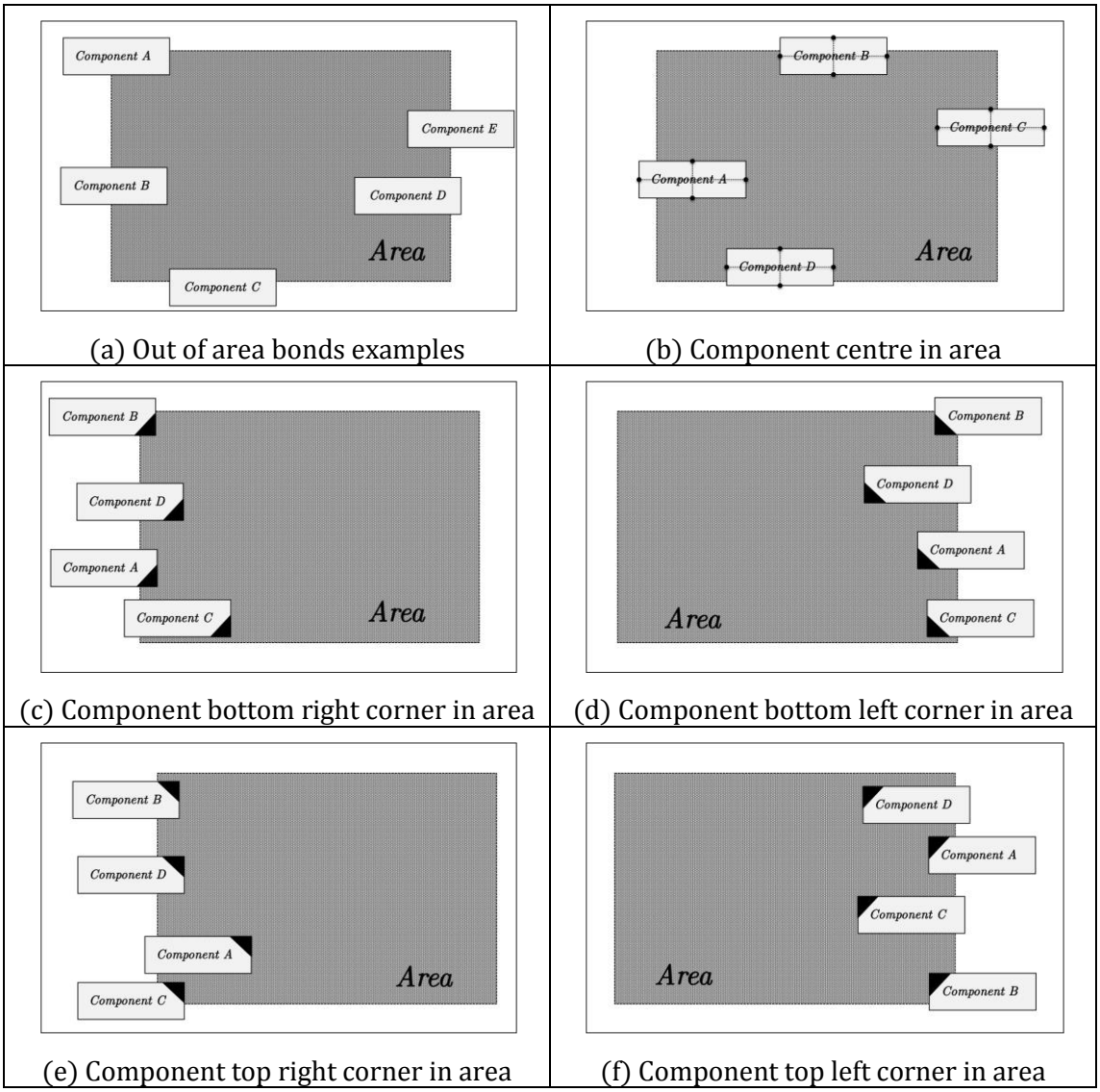



Figure 69: Out of area scenarios

Toward the end of the GA run, the main selective pressure mostly takes into account the human factors during its selection rounds leading the GA to produce layouts that are more usable in terms of usability.

## 5.10 Lisp Version design

The Lisp version of TOISE (Its design can be seen in Figure 41) relied on CFFI (Lisp side) & SWIG (SWIG 2012) (Native code side) code wrappers to communicate bi-directionally with the native code held in the library DLL that also contain the GUI and most of the functionality. Most of the methods that are used in the code are callbacks as the communication originates from the GUI held in the DLL to pass commands to ACT-R or Lisp which most of the time expects some kind of return values. Thus the order of loading the individual elements in the system is important. First ACT-R is loaded, then its

environment, followed by the CFFI backend, which will provide the interface to the methods help in the library once it is loaded. This is followed by both the methods and callbacks. Finally, a method held in the library is invoked to start the GUI. The loading process is described in Figure 70.

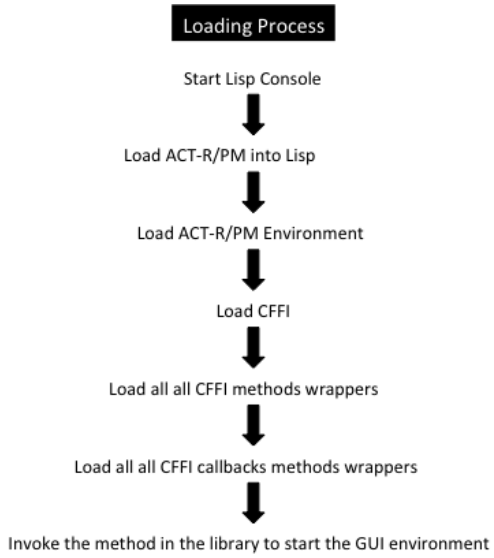



Figure 70: Loading the project from Lisp

Unlike the code in the Lisp environment, which is not compiled, the C++ native code side is double-compiled. The first compilation gets SWIG to generate all the wrapper code necessary for the library methods that are exported and contained in the DLL. It also generates the Lisp code based on CFFI to integrate and load in the Lisp environment. The second compilation compiles the code wrappers with the library code so that the methods shared between CFFI and the DLL can be assessed by ways of "bindings" without regard to provenance.

In the DLL code, a special header is thus needed to let SWIG know which methods must be included in the wrapping exercise. This is often contained in a header file with the "i" extension. An example of such as file is presented in Figure 71.

```
%module LispDll
%{
  #include "stdafx.h"
  #include "LispDll.h"

%}
%include "LispDll.h"
```

Figure 71: Typical SWIG header file for native code

In the header Swig file shown in Figure 71, we simply tell SWIG to refer to the "*LispDLL.h*" DLL library header file and export everything that is declared.

```
#define _CRTDBG_MAP_ALLOC
#define _CRTDBG_MAPALLOC
#include <stdlib.h>
#include <crtdbg.h>
#pragma once
#ifndef _LispDll
  #define _LispDll
  #ifdef __cplusplus
    extern "C" {
  #endif
     .........

    void setLoadModelFunction(void(*loadModel)(const char*));
    void setRunModelFunction(void(*runModel)(int));
    void startDialog(void);
    void endDialog(void);
    void setCreateIDTRHooks(void (*createIDTR)());
    void setDeleteIDTRHooks(void (*createIDTR)());
      .........

  #ifdef __cplusplus
    }
  #endif
#endif
```

Figure 72: Typical DLL header file

A typical header file is shown in Figure 72. An example of the SWIG wrappers is shown in Figure 73.

```
EXPORT void _wrap_setLoadModelFunction (void (*larg1)(char const *)) {
  void (*arg1)(char const *) = (void (*)(char const *)) 0 ;
  arg1 = larg1;
  try {
    setLoadModelFunction(arg1);

  } catch (...) {

  }
}

EXPORT void _wrap_setRunModelFunction (void (*larg1)(int)) {
  void (*arg1)(int) = (void (*)(int)) 0 ;
  arg1 = larg1;
  try {
    setRunModelFunction(arg1);

  } catch (...) {

  }
}

EXPORT void _wrap_startDialog () {
  try {
    startDialog();

  } catch (...) {

  }
}

EXPORT void _wrap_endDialog () {
  try {
    endDialog();

  } catch (...) {

  }
}
```

Figure 73: SWIG Native code wrappers

The names of the methods that are wrapped are similar to the methods SWIG generates for the CFFI wrapping code that is used on the Lisp side as seen in Figure 74.

```
(cffi:defcfun ("_wrap_setLoadModelFunction" setLoadModelFunction) :void
(loadModel :pointer))

(cffi:defcfun ("_wrap_setRunModelFunction" setRunModelFunction) :void
(runModel :pointer))

(cffi:defcfun ("_wrap_startDialog" startDialog) :void)

(cffi:defcfun ("_wrap_endDialog" endDialog) :void)
```

Figure 74: Lisp CFFI wrapper code generated by SWIG

Once the final compile is processed, the DLL is loaded from Lisp and both environments can invoke each other methods as being part of the same system.

## 5.11 Issues with the TOISE development.

There were many issues regarding the Lisp version of TOISE system in terms of communication between the two main parts. Firstly the dual communication between modules had to be extra fast so Lisp would not crash due to the pressure the GA contained in the DLL puts on the Lisp backend. This entailed independent and fast threading algorithms to deal with both ways communication. Secondly, the DLL library had to be loaded by the Lisp environment and had its exposed functionality usable from Lisp and vice versa. This functionality was provided by the use of CFFI. With this design, the former provided an interface to ACT-R to the GA installed in a DLL, which was somewhat limited. The latter provided an interface to foreign function code to native code, allowed the allocation and dereferencing for foreign memory, invoking of foreign methods, including callbacks that is shared in libraries from a Lisp environment. It also provided support to load these libraries into the Lisp environment.

This was compounded by the fact that the GA is highly iterative thus putting pressure on both systems without giving them time to, for instance, clean the memory up. This matter was not a problem with the code in the library but was at the level of Lisp, which uses a Garbage Collector, which seemed to respond too slowly at deleting resources thus making the system consuming more memory than needed.

As already mentioned, the Lisp version had two main parts. The main part in terms of the design was the Lisp Environment in which ACT-R was loaded and ran. Swig and CFFI were used to provide wrappers and functionality to load Native code libraries i.e. DLL's for Windows and to make code from both environment available to each other using wrapper code.

Each communication channels that allowed data to flow from ACT-R and the GA relied entirely on independent threads, as it was found that locking code i.e. using threading locks, between channels often resulted in the Lisp console freezing. Thus special care was used to develop optimised code that recycles the data as fast as possible so to minimize possibilities of crashes. Moreover it allowed the free manipulation of all controls in the GUI contained in the DLL, which would not have been possible once the GA underwent a run for instance.

It was furthermore found that due to the way Swig worked at the time of development i.e. Swig making copies of data structures to similar structures that are suitable with the intended receiving environment, small memory leaks could happen. The main issues in

that respect were that even if special care was used to delete and manage resources in the DLL native code, due to the combination of the use of the garbage collector which only delete resources over a number of lives and internal leaks in the SWIG wrappers, loss of memory would occur over a long period of time. As a genetic algorithm is highly iterative, a small leak over one GA run generation might be multiplied a million of times, the overall memory that is not recycle can affect the running of the system. Moreover this was compelled further by the fact that the pressure the GA module puts on the Lisp environment, the Lisp garbage collector was often interrupted and too slow at deleting unneeded resources. Special care was therefore taken to develop an external thread in Lisp that instigated and pushed the garbage collector to be more assertive in deleting unwanted resources, which kept resources loss at its minimum.

## 5.12 TOISE ACT-R code recompiled in C++

The Lisp version suffered many issues that were difficult to resolve; the major issue being the speed of execution due to the speed of execution of ACT-R Lisp environment. For this reason part of the code contained in ACT-R was ported and optimised in the C++ library to increase the speed of execution. In the same way Salvucci did with some experimentation (Myers 1993; Salvucci 2001). There is an increasing trend to use accurate simulation code in the design of many systems to improve their reliability. These simulations in terms of computations and process timing are often overwhelmingly time consuming. This is often a compelling side in optimisation algorithms using these types of high quality, reliable simulations for fitness assessment purposes. Alongside this trend, system designers are now designing low cost fitness surrogate functions to replace high computationally expensive simulation codes (Lim et al. 2007; Zhou et al. 2007; Couckuyt et al. 2011).

This was the case with the Lisp version of TOISE. The high cost of individual evaluations shaded doubts to its usability. This led to a new approach, which included at its hart the compilation of the relevant ACT-R code into the GA (C++) base native code. Some to the source code is listed in Appendix J, which include some of the functionality pertaining to dragging an object. A lot of the methods listed are *inline* meaning that they are speed optimised, using also pointers as much as possible to reduce object creations etc. Some of the functionality was a straight port from ACT-R whilst others were achieved by inspecting carefully some debug logs that positioned throughout the source code on both C++ and Lisp platforms leading to corrective tweaks in the C++ native to realign results to push them in line with ACT-R findings. This was a long and tedious process

involving a lot of statistical tests which was only possible due the small granularity nature of the actions modelled.

To validate the solution, a comparison was made between ACT-R latencies and the simulated ones for a full run where ACT-R and the simulator returned latencies for the same tasks. Over 100000 comparisons were made in this run. This technique is often used in optimisation using GA where often computer intensive modelling code is replaced by simulation for fitness assessment of the individuals of the GA population. Perusing both Table 13 and Table 14, the simulator does not induce major calculations errors given the fact of the precision of the number used.

Table 13: ACT-R versus Simulator

| | Mean | Std. Deviation | N |
|---|---|---|---|
| **ACT-R** | 77.1487609 | 0.57902102 | 101648 |
| **Simulator** | 77.1488846 | 0.57897879 | 101648 |

Moreover, these slight differences in terms of value should not affect the GA ability to evolve any of those solutions accordingly as we will see in later sections, ACT-R latency is only one part of the whole fitness value; the others being the different layout penalties and constraints that are equally important for most of a GA run.

Table 14: ACTR versus Simulator correlation

| | | ACT-R | Simulator |
|---|---|---|---|
| **ACT-R** | Pearson Correlation | 1 | 1.000** |
| | Sig. (2-tailed) | | .000 |
| | Sum of Squares and Cross-products | 34078.717 | 34074.521 |
| | Covariance | .335 | .335 |
| | N | 101648 | 101648 |
| **Simulator** | Pearson Correlation | 1.000** | 1 |
| | Sig. (2-tailed) | .000 | |
| | Sum of Squares and Cross-products | 34074.521 | 34073.746 |
| | Covariance | .335 | .335 |
| | N | 101648 | 101648 |

**. Correlation is significant at the 0.01 level (2-tailed).

The porting of the ACT-R code into the GA environment makes possible long runs with runs times in the range of hours compared to days using ACT-R in the Lisp environment.

# Chapter 6. Experimentation

There are two classes of search algorithms: Global and local search algorithms. The former algorithms, which include genetic algorithms, are characterized by finding the right equilibrium between exploration of the search space and exploitation of accumulated information to help the search process to find the best global solution. Probabilistically, these algorithms are guaranteed to find a global solution at the expense of long computations. However given their robust nature, global algorithm can explore the search space and locate reasonably good global solutions relatively fast. The latter type of algorithms on the other hand does not rely on exploration in the same manner as global algorithms do to locate good solutions but move from a current solution to a nearby-improved solution present in the neighbourhood often terminating when there is no longer any improvement that can be discovered. The weakness with these types of algorithms is two folds: depending on the complexity of the search space and the initial solution that started the search process, they can focus on local optima solutions, and finally they are often associated with much longer run times than global search algorithms (Balaprakash 2012).

It is therefore of interest to determine if the global genetic search algorithm contained in TOISE competes with a local search algorithm that inherits the same shift functionality and to determine if the finding retrieved during these experiments discussed in the following sections ascertain if the case for using a genetic algorithm is founded.

A lot of evaluations have already been discussed in the sections above. However there is a need for more systematic experimentation when dealing with both human factors and genetic algorithms. Whilst for the former one can rely on already published literature as the system has stayed closely similar in terms of methodology to many published research in this domain i.e. GOMS to ACT-R5 using G2A, and then translation from ACT-R5 to ACT-R6 with minor platform adjustments, the latter is somewhat more challenging to assess its performance and therefore the overall ability of the system (that is using ACT-R as part of its fitness assessment) to find competitive optimised layout in a reasonable amount of time.

The first set of experiments re-examine findings published in WCCI IEEE publication (Golovine et al. 2010), based on real world problem (Deininger 1960). The main purpose of this series of experiment was to re-discover some of the findings that Bell Laboratories achieved at much expense and time. Bell laboratories evaluated 16 different designs using

more than 200 participants and the research lasted two years and came up with the now ubiquitous digital phone keypad layout that is part of most of the communication devices today. The approach presented in this thesis reliably produced similar competitive layouts to Bell's, all of them better in terms of human factors than the one proposed by Bell's. As with all human interaction, small differences are expected but are significant in terms of cognitive load when the apparatus is user over a long period of time. This set of experiments will attempt to determine if a local search based on the genetic algorithm can perform as well as the genetic algorithm and if these results compared to already published ones.

The second set of experiment re-examine the calculator software that was used to demonstrate the software in the previous chapter. In these experiments more constraints and human interaction are used to provide additional pressure and demonstrate the concept behind TOISE.

### 6.1.1 Experiment 1

#### 6.1.1.1 Aims and objectives

The aim of this experiment is to define if both algorithms can find competitive layout solutions given two different population diversities and relatively low number of individuals for the genetic algorithm i.e. 25, using the real world problem first defined by Bell Laboratories (1960) (Golovine et al. 2010). In this experiment 10 phones numbers were used.

#### 6.1.1.2 Experiment setup.

The GA is setup with a population comprising 25 individuals, running for 10000 generations, with a base crossover set at 0.7. As it is an automatic process both the mutation and crossover rate will fluctuate once the GA is running. The pre-run found mutation rate is passed over to the LS algorithm so both are in par, with the LS running for 25*10000 iterations. (No crossover exists for this algorithm). This rate was found to be 0.009 for a population diversity = 0.001 and 0.03 for a diversity of 0.01. Two series of experiments were setup to run for both population diversities. The interface is shown in Figure 75.

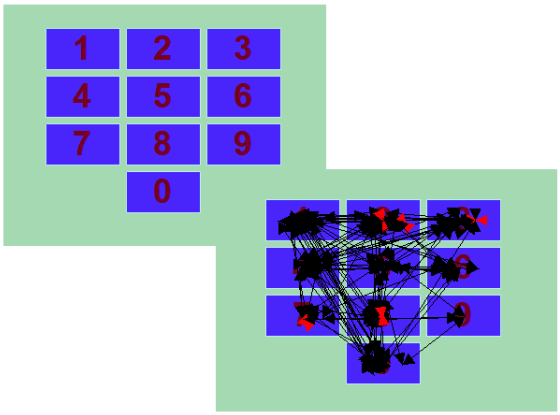



Figure 75: 10 Phone numbers interaction

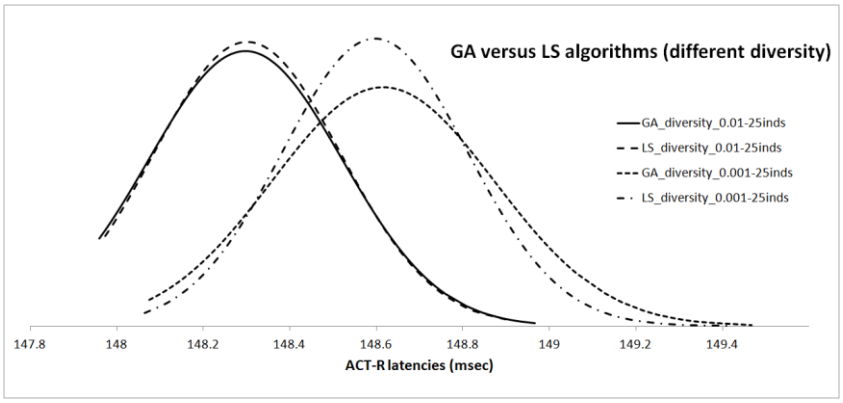



Figure 76: GA versus LS (Different population diversity – 25 individuals)

#### 6.1.1.3 Experiment results

Perusing Table 15, both GA and LS approach are capable to find layouts as good or better as the original layout. i.e. <= 151.737. Looking closer at Figure 76, one can see that all layouts found by both methods are better in terms of ACT-R latencies than the one obtained using the original layout and therefore both algorithms are capable to find competitive layouts regardless of the population diversity (hence mutation rate). Referring to Table 15 and Figure 76, the obtained means show that within their own category i.e. diversity = 0.01 or 0.001, the distribution curves are very similar and none of the approach seems better than the other with no significant difference in the findings i.e. (t (493.930) =-1.490, p = .137) for population diversity set to 0.001 and (t (498) = -.459, p .646) when set to 0.01. It is worse noting that the LS type of algorithm found the best layout i.e. ACT-R latency = 147.9740, for a population diversity set to optimised 0.1 % compared to the LS ACT-R latency = 147.9600. It is however a very small difference but still is significant in terms of cognitive load.

#### 6.1.1.4 Discussion

The final point to note is that the GA runs lasted only hours whereas the LS algorithm ran for many days i.e. around 3 days to complete the 250 runs. This is due to the parallel architecture that is embedded into a genetic algorithm, thus making them much more efficient that the purely sequential approach that forms the basis for a local search type of search algorithm. The findings demonstrate that given the mutation algorithms (that shifts the components of the layout around) that were shared by both the LS and GA are producing layouts that can provide any approaches to produce competitive layouts. Finally, the small differences found for the best ACT-R latencies between the two approaches are not significant in terms of cognitive load i.e. 4ms. It is also worth mentioning that the optimal gene mutation rate for the LS approach was discovered by the GA approach.

Table 15: GA versus LS (Diversity=0.001-25 individuals)

| Algorithm | N | Minimum | Maximum | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|---|
| [1]: GA (diversity=0.001) | 250 | 148.0750 | 149.4670 | 148.615380 | .2572204 | .0162680 |
| [2]: LS (Mutation rate = 0.009) | 251 | 148.0000 | 149.4200 | 148.651590 | .2858941 | .0180455 |
| [3]: GA (diversity=0.01) | 250 | 147.9600 | 148.9660 | 148.297560 | .2232707 | .0141209 |
| [4]: LS (Mutation rate = 0.03) | 251 | 147.9740 | 149.2900 | 148.306096 | .2247648 | .0141870 |

| T-Test | | | | | |
|---|---|---|---|---|---|
| Test type | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference |
| [1] & [2] | -1.490 | 493.930 | .137 | -.0362096 | .0242959 |
| [3] & [4] | -.459 | 498 | .646 | -.0092000 | .0200460 |
| [1] & [3] | -14.754 | 498 | .000 | -.3178200 | .0215418 |
| [2] & [4] | -15.006 | 473.662 | .000 | -.3448296 | .0229800 |

### 6.1.2 Experiment 2

#### 6.1.2.1 Aims and objectives

The aim of this experiment is to define if both algorithms can find competitive layout solutions given two different population diversities, scaling up the number of individuals for the genetic algorithm i.e. 50, using the same real world problem as discussed in previous experiment with the same number of phone numbers i.e. 10.

#### 6.1.2.2 Experiment setup.

The GA is setup with a population comprising 50 individuals, running for 10000 generations, with a base crossover set at 0.7. As it is an automatic process both the mutation and crossover rate will fluctuate once the GA is running. The pre-run found mutation rate was passed over to the LS algorithm so both are in par, with the LS running for 50*10000 iterations. (No crossover exists for this algorithm). This rate was found to be

0.006 for a population diversity = 0.001 and 0.02 for a diversity of 0.01. Two series of experiments were setup to run for both population diversities.

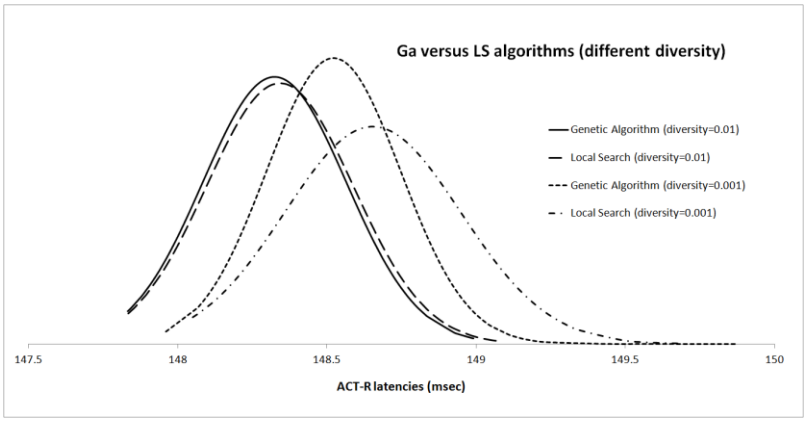



Figure 77: GA versus LS (Different population diversity – 50 individuals)

#### 6.1.2.3 Experiment results

Perusing Table 16, both GA and LS approach are capable to find layouts as good or better as the original layout. i.e. <= 151.737. Looking closer at Figure 77, one can see that all layouts found by both methods are better in terms of ACT-R latencies than the one obtained using the original layout and therefore both algorithms are capable to find competitive layouts regardless of the population diversity (hence mutation rate). Referring to Table 16 and Figure 77, the obtained result means show a slight advantage towards the GA when the mutation rate is not optimised ($t(499) = 2.945$, $p = .003$). However this difference between those means disappears and becomes not significant as the mutation rate becomes more optimised for the solution ($t(499) = .99$, $p = .323$). It is worse noting that the GA type of algorithm found the best layout i.e. ACT-R latency = 147.832, for a population diversity set to optimised 0.1 % compared to the LS ACT-R latency = 147.834.

#### 6.1.2.4 Discussion

The final point to note is that the GA runs lasted only hours whereas the LS algorithm ran for many days i.e. 5 days to complete the 250 runs. As already mentioned, the parallelism of GA makes this approach much more efficient than a local search approach. Another point to note is that unlike the experiment with 25 individuals, when the Local search approach was better than the GA when the mutation rate was not optimised, doubling the number of individuals has for effect that the GA becomes more efficient when running with non-optimal population diversity which seems to indicate that the extra diversity provided by the increase number of individuals benefits the GA more than it does for a LS approach.

| Table 16: GA versus LS (50 inds-10 numbers) | | | | | | |
|---|---|---|---|---|---|---|
| **Algorithm** | **N** | **Minimum** | **Maximum** | **Mean** | **Std. Deviation** | **Std. Error Mean** |
| **[1]: GA (diversity=0.001)** | 250 | 147.9600 | 149.8780 | 148.580640 | .2823018 | .0178543 |
| **[2]: LS (Mutation rate = 0.006)** | 251 | 148.0500 | 152.1690 | 148.667016 | .3684172 | .0232543 |
| **[3]: GA (diversity=0.01)** | 250 | 147.8320 | 148.9900 | 148.323772 | .2401823 | .0151905 |
| **[4]: LS (Mutation rate = 0.02)** | 251 | 147.8340 | 149.1010 | 148.345446 | .2498718 | .0157718 |

| **T-Test** | | | | | |
|---|---|---|---|---|---|
| **Test type** | **t** | **df** | **Sig. (2-tailed)** | **Mean Difference** | **Std. Error Difference** |
| [1] & [2] | 2.945 | 499 | .003 | .0863759 | .0293332 |
| [3] & [4] | .990 | 499 | .323 | .0216742 | .0218992 |
| [1] & [3] | -10.958 | 485.541 | .000 | -.2568680 | .0234420 |
| [2] & [4] | -11.444 | 439.831 | .000 | -.3215697 | .0280982 |

### 6.1.2.5 Summary

Looking at data distribution curves displayed in Figure 78, one realises that once the population diversity is set appropriately, changing the number of individuals from 25 to 50 does not make any changes in the behaviour of both approaches as both methods are able to find layouts better than the original ones providing evidence that maintaining the population diversity in a search algorithm of this type is paramount.

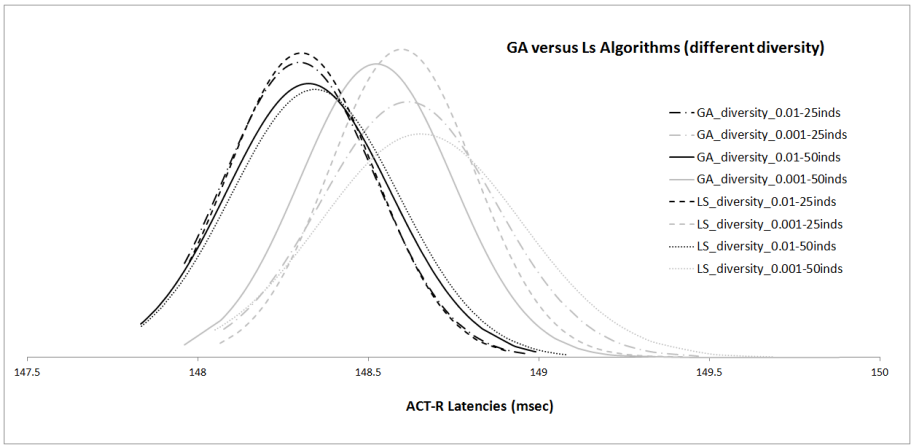



Figure 78: 10 phones - (25/50 Individuals) - Difference diversities

## 6.1.3 Experiment 3

### 6.1.3.1 Aims and objectives

The aim of this experiment is to determine if scaling up the number of phone numbers to 100 phone numbers makes a difference in the behaviour of both approaches using the Bell layout problem. The assumption is that the increase of activity on the layout should help both approaches to find better high quality by increasing the interaction pressure of the algorithms.

### 6.1.3.2 Experiment setup.

The experiment setup is the exactly the same as the previous experiment with only the number of interaction with the keypad changed to 100 phone numbers. The pre-run found mutation rate was passed over to the LS algorithm so both are in par, with the LS running for 25*10000 iterations. (No crossover exists for this algorithm). This rate was found to be 0.01 for a population diversity = 0.001 and 0.03 for a diversity of 0.01. Two series of experiments were setup to run for both population diversities.

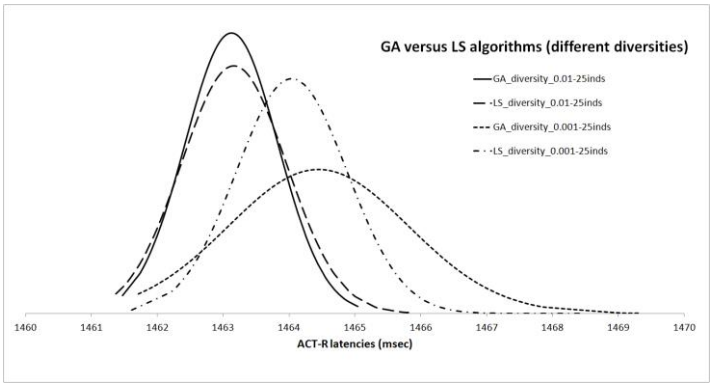



Figure 79: GA versus LS - 100 phones-25 individuals

### 6.1.3.3 Experiment results

Perusing Table 17, both GA and LS approach are capable to find layouts as good or better as the original layout i.e. < 1487.481. Looking closer at Figure 79, one can see that all layouts found by both methods are better in terms of ACT-R latencies as demonstrated by the previous experiments.

Table 17: GA versus LS (25 inds-100 numbers)

| Algorithm | N | Minimum | Maximum | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|---|
| [1]: GA (diversity=0.001) | 250 | 1461.7100 | 1469.3400 | 1464.445200 | 1.3678982 | .0865135 |
| [2]: LS (Mutation rate = 0.01) | 251 | 1461.6100 | 1468.6700 | 1464.322231 | 1.1742493 | .0741180 |
| [3]: GA (diversity=0.01) | 250 | 1461.4700 | 1465.0400 | 1463.122960 | .7025708 | .0444345 |
| [4]: LS (Mutation rate = 0.03) | 251 | 1461.3700 | 1466.1400 | 1463.172590 | .8164876 | .0515362 |

| T-Test | | | | | |
|---|---|---|---|---|---|
| Test type | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference |
| [1] & [2] | 1.079 | 487.230 | .281 | .1229689 | .1139213 |
| [3] & [4] | -.729 | 499 | .466 | -.0496296 | .0680674 |
| [1] & [3] | -13.595 | 371.824 | .000 | -1.3222400 | .0972574 |
| [2] & [4] | -12.735 | 445.939 | .000 | -1.1496414 | .0902743 |

Referring to Table 17 and Figure 79, the obtained result means dismiss any significant advantage between any approaches when the mutation rate is not optimised (t (487.230) = 1.079, p = .281). The same findings are obtained when the mutation rate becomes more optimised for the solution (t (499) = -.729, p .466). It is worse noting that the LS type of

algorithm found the best layout i.e. ACT-R latency = 1461.3700, for a population diversity set to optimised 0.01 compared to the GS ACT-R latency = 1461.4700.

#### 6.1.3.4 Discussion

The findings obtained in this experiment are consistent with the previous experiments results. When the GA population size is set to 25 individuals, scaling up the user interaction on the interface from 10 to 100 phone numbers does not make significant changes in terms of differentiating both methods. The only major issues experienced with the LS method is again the time taken to generate any solution which increases substantially for the LS approach, a difference that is in a range of hours for the GA to days for the LS method. It is worth mentioning that both presented systems in this thesis (i.e. GA and LS) performed far better than the ones that were the subject of investigation in (Golovine et al. 2010). Similarly to those published results, both LS and GA have found better keypad layouts in terms of human interaction latencies than the original keypad presented in Deininger (1960) report (Deininger 1960).

### 6.1.4 Experiment 4

#### 6.1.4.1 Aims and objectives

The aim of this experiment is to determine if scaling up the population size makes a difference if the behaviour of both approaches using the Bell layout problem. The assumption is that the increase of the number of individuals should help both approaches to find high quality by increasing the interaction pressure of the algorithms. The difference with the previous experiment is that the number of individuals of the population is increased from 25 to 50 individuals.

#### 6.1.4.2 Experiment setup.

The experiment setup is the exactly the same as the previous experiment with the number of interactions with the key pad remaining 100 phone numbers but the population size is increased from 25 to 50 individuals. The pre-run found mutation rate was passed over to the LS algorithm so both are in par, with the LS running for 25* 10000 or 50*10000 iterations. (No crossover exists for this algorithm). This rate was found to be 0.02 for a diversity of 0.01. Two series of experiments were setup to run for both population diversities.

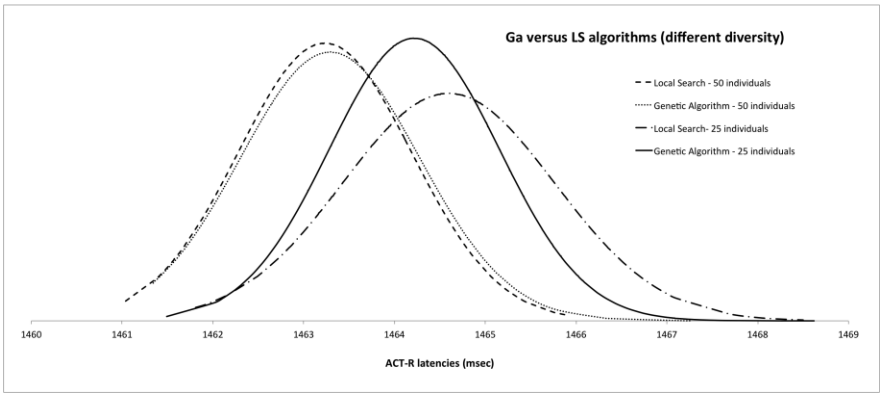



Figure 80: GA versus LS - 100 phones-25 and 50 individuals' population

#### 6.1.4.3 Experiment results

Perusing at Table 18 both GA and LS approach are capable to find layouts as good or better as the original layout in the same manner regardless if the population size is set to 25 or 50 individuals. Looking closer at Figure 80, one can see also that all layouts found by both methods are better in terms of ACT-R latencies as demonstrated by the previous experiments. Referring to Table 18 and Figure 80, the obtained result means show a significant advantage for both methods when the population is raised from 25 to 50 individuals ($t\ (475.364) = 12.093$, $p = 0.000$) and ($t\ (481.013) = 14.075$, $p = 0.000$) for the GA and LS respectively. However comparing GA and LS using 50 individuals i.e. longer search times, the difference is not significant ($t\ (499) = 0.515$, $p = 0.607$) with similar findings for shorter run times i.e. using 25 individuals ($t\ (499) = -1.047$, $p = 0.295$).

#### 6.1.4.4 Discussion

Increasing the population size from 25 to 50 individuals does make a significant difference in terms of performance for both approaches in terms of ACT-R latencies. However comparing the two approaches for the two setting of population size does not provides any evidence that one approach performs better than the other. Similar to all other experiments, the run time for the LS algorithm was much more time consuming that it was for the GA method.

| Table 18: versus LS (50 inds-100 numbers) | | | | | | |
|---|---|---|---|---|---|---|
| **Algorithm** | **N** | **Minimum** | **Maximum** | **Mean** | **Std. Deviation** | **Std. Error Mean** |
| [1]: Genetic Algorithm - 25 individuals | 250 | 1461.49 | 1468.62 | 1464.49812 | 1.232195 | 0.0779309 |
| [2]: Local Search - 25 individuals | 251 | 1461.81 | 1468.56 | 1464.611474 | 1.1902927 | 0.0751306 |
| [3]: Genetic Algorithm - 50 individuals | 250 | 1461.34 | 1467.26 | 1463.2906 | 0.9870979 | 0.0624296 |
| [4]: Local Search - 50 individuals | 251 | 1461.04 | 1466.33 | 1463.245498 | 0.9732079 | 0.0614283 |

| T-Test | | | | | |
|---|---|---|---|---|---|
| **Test type** | **t** | **df** | **Sig. (2-tailed)** | **Mean Difference** | **Std. Error Difference** |
| [1] & [2] | -1.047 | 499 | 0.295 | -0.1133541 | 0.1082414 |
| [3] & [4] | 0.515 | 499 | 0.607 | 0.045102 | 0.0875811 |
| [1] & [3] | 12.093 | 475.364 | 0.000 | 1.20752 | 0.0998532 |
| [2] & [4] | 14.075 | 481.013 | 0.000 | 1.3659761 | 0.0970466 |

### 6.1.4.5 Discussion

Increasing the population size from 25 to 50 individuals does make a significant difference in terms of performance for both approaches in terms of ACT-R latencies. However comparing the two approaches for the two setting of population size does not provides any evidence that one approach performs better than the other. Similar to all other experiments, the run time for the LS algorithm was much more time consuming that it was for the GA method.

## 6.1.5 Experiment 5: Optimising a calculator Interface

The purpose of this experiment is to take the calculator that was already used to discuss the development of TOISE and demonstrate some of the powerful search of the TOISE system. The interaction recording was achieved using the ROBOT recorder where 146 actions were performed on the calculator as shown in Figure 81.

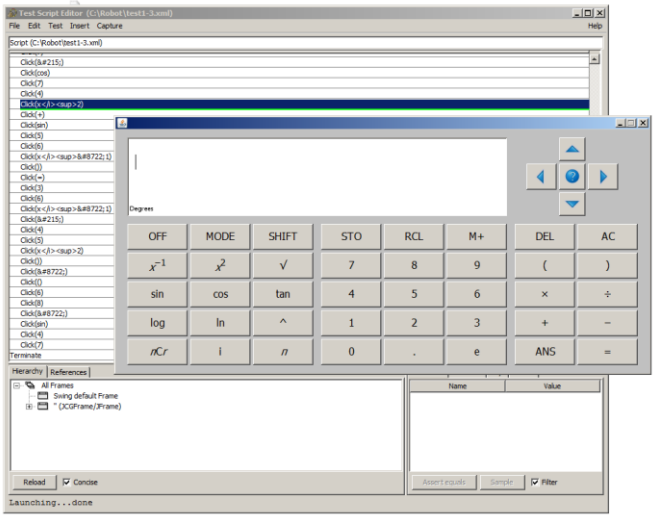



Figure 81: Calculator Experiment conducted using the ROBOT recorder

The first experiment considers the original design where three areas are added and the components that were part of this area in the original design are bound to designed areas. Some components at the top of the application are the display component and control components but were not used when exercising the software using the ROBOT. They are shown using the yellow colour, which indicates that the TOISE will not attempt to shift them during its runs. It was also decided for this run to have the keyboard keys remaining static as well. This is an example where an already designed interface exists, and an attempt to optimise some related components is made around the static components that cannot be shifted. This interface is shown in Figure 82. User interaction is shown in Figure 83.

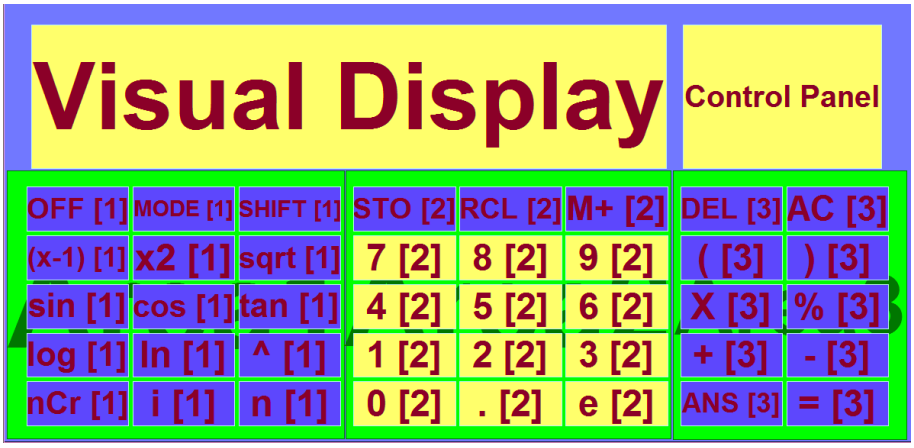



Figure 82: Experiment with three areas designed

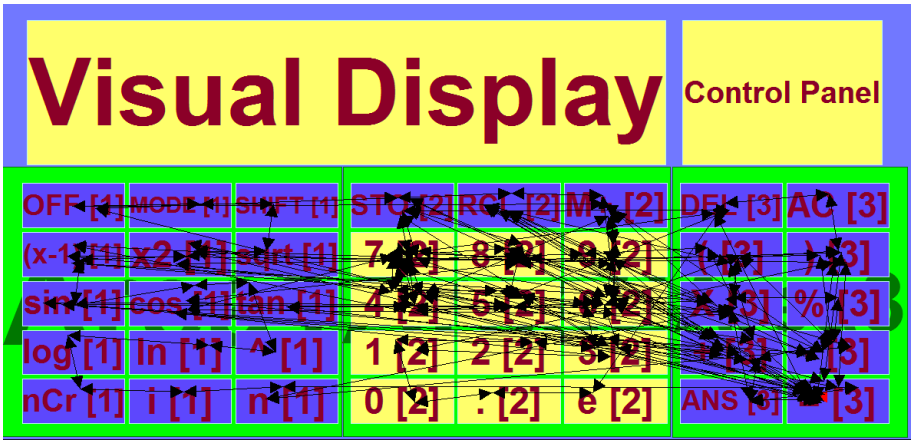



Figure 83: Experiment with three areas designed (with user interaction)

The other experiment seen in Figure 84 conducted adds another dimension of difficulty by adding another extra area. Unlike the previous tests where components were left loose in any areas, this constrains the *OFF*, *MODE* and *SHIFT* function keys to remain in small area just below the *Visual Display* area. The other major difficulties are the constraining of the *STO*, *RCL*, *M+*, *DEL* and *AC* function key to another area. The difficulties stem from the

fact that the memory storage key *STO*, the recall key *RCL* and the key to add the answer to the memory storage *M+* are particularly tied with the + and = keys (see Figure 85) which has the effect to draw those keys very close to each other to minimise the interaction latency. Therefore the penalty associated with *out of area* and *overlapping* was raised from the standard 5000, to the more appropriate 50000 to put more pressure regarding the integrity of the layout. Hence, once the GA find a proper layout for the interface that does not breach these rules, the focus is drawn on the user interaction within those areas.

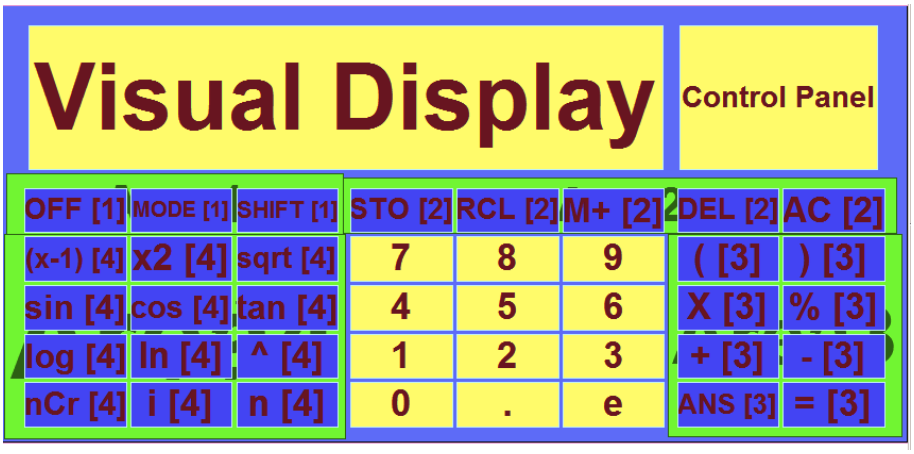



Figure 84: Experiment using four areas to constrain components

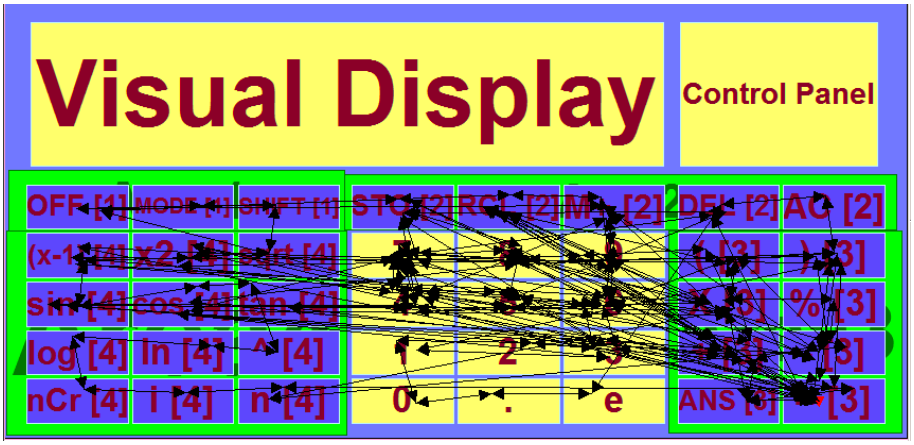



Figure 85: Experiment using four areas to constrain components (with Actions)

A standard setting is used otherwise i.e. 50 individuals for this experiment. Some tests were previously conducted to assess the population diversity using the simple user interaction recording (41 interactions used only for run speed). 250 runs were conducted for the diversity tests included a range of diversity values. The distribution curves for the series of experiments are shown in Figure 86 . One can infer from these findings that any diversity values ranging from 0.04 to 0.1 do produce similar results. Thus 0.1 was used to conduct the experiments as this might push the GA to discover layouts that maybe come below the two standard deviations threshold i.e. only diversity = 0.1 and 0.05 tests

contains more than one values below the two standard deviations. Looking at Table 19, both perform similarly (t (50) = -.369, p = .713) but the skewness and kurtosis for the diversity = 0.1 is smaller. In addition, this higher population diversity can be beneficial in a complex problem as this.

The crossover for the main experiment was left to its standard value of 0.7 and each run ran for 5000/10000/20000 generations. Around 35.250 billion user actions will be assessed during these tests i.e. 50*10000*141*250.

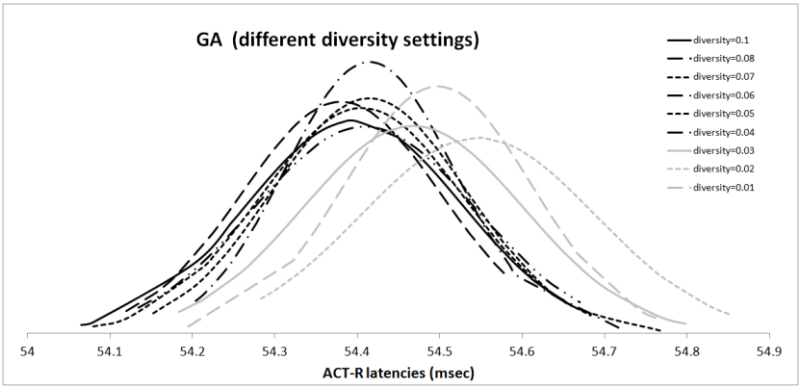



Figure 86: Performance for difference diversities

Table 19: Diversities comparison

| | N | Minimum | Maximum | Mean | | Std. Deviation | Variance |
|---|---|---|---|---|---|---|---|
| **Type** | **Statistic** | **Statistic** | **Statistic** | **Statistic** | **Std. Error** | **Statistic** | **Statistic** |
| **Diversity=0.1** | 50 | 54.06480 | 54.65870 | 54.39645 | 0.01850 | 0.13082 | 0.01711 |
| **Diversity=0.05** | 50 | 54.08000 | 54.68310 | 54.40582 | 0.01743 | 0.12324 | 0.01519 |
| **Skewness and Kurtosis** | | | | | | | |
| | | **Skewness** | | | **Kurtosis** | | |
| | | **Statistic** | **Std. Error** | **Z Value** | **Statistic** | **Std. Error** | **Z Value** |
| **Diversity=0.1** | 50 | -0.17367 | 0.33660 | 0.51595 | -0.00444 | 0.66191 | 0.00671 |
| **Diversity=0.05** | 50 | -0.26856 | 0.33660 | 0.79787 | 0.30227 | 0.66191 | 0.45666 |
| **t-test for Equality of Means** | | | | | | | |
| | | **t** | **df** | **Sig. (2-tailed)** | **Mean Difference** | **Std. Error Difference** | |
| | | -.369 | 98 | .713 | -.0093760 | .0254178 | |

### 6.1.5.1 Experiment results

The first noticeable fact that can be seen looking at the distribution curves for the obtained data is that adding extra areas does impact on the quality of the solutions found in terms of human cognitive and motor load that has to be achieved to complete this sequence of actions using this interface.

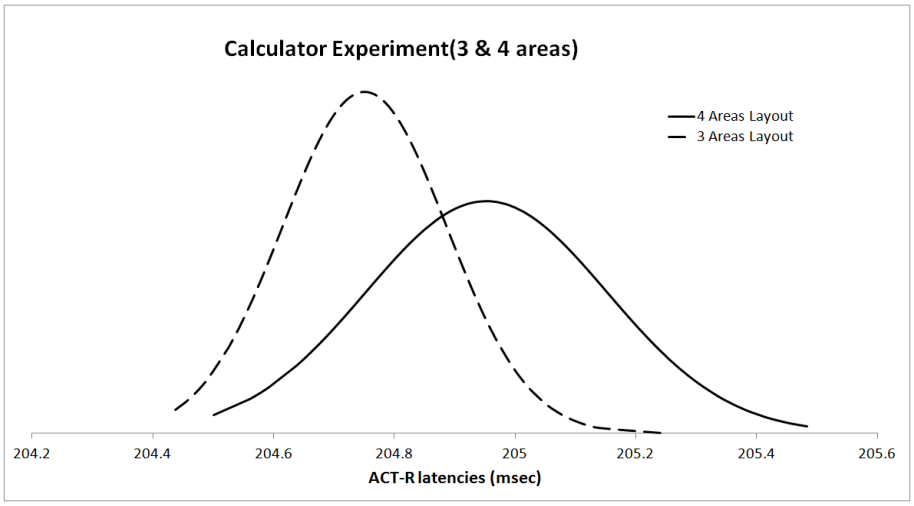



Figure 87: Distribution curves for Calculator experiments (10000 generations)

This is re-enforced by the conducted T-Test shown in Table 20 ($t(438.327) = -13.184$, $p = .000$), which shows that using only three areas produces better ACT-R latencies.

Table 20: Comparing Calculator experiments (10000 generations)

| Algorithm | N | Minimum | Maximum | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|---|
| [1]: 4 Areas Experiments | 250 | 204.5010 | 205.4830 | 204.952256 | .2000618 | .0126530 |
| [2]: 3 Areas Experiments | 250 | 204.4370 | 205.2450 | 204.750628 | .1358289 | .0085906 |

| T-Test | | | | | |
|---|---|---|---|---|---|
| **Test type** | **t** | **df** | **Sig. (2-tailed)** | **Mean Difference** | **Std. Error Difference** |
| [1] & [2] | -13.184 | 438.327 | .000 | -.2016280 | .0152937 |

To assess the results the top 5% solutions as well as the solutions with an ACT-R latency below two standard deviations were taken into account. Both approaches found two solutions that were below the two standard deviations threshold.

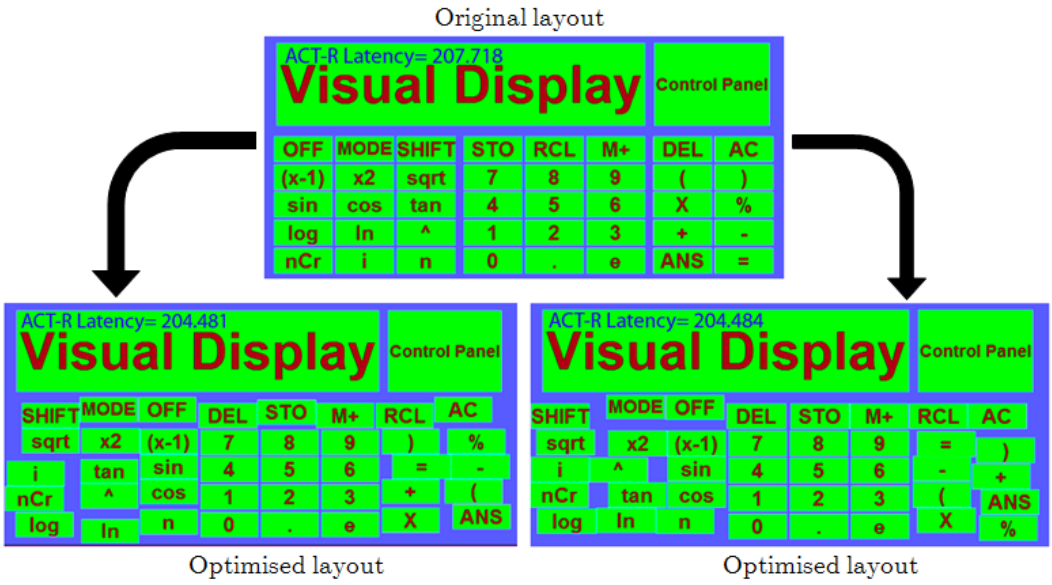



Figure 88: Top two optimised (< 2* STDEV) layouts
using (4 areas) 10000 generations

Perusing Figure 88, the optimised layouts using the *four areas* approach design suggests similar arrangements in many areas. The area on the left of the keypad is arranged differently though despite containing many similarities. Both top areas that contain the function keys have been laid out differently from the original to optimise the user interaction in a similar fashion. The right area is also constant from for both layouts but does contains some slight differences i.e. ^ and *TAN*. The optimised layouts provide a gain of over 3 seconds in terms of ACT-R timings, which is a significant gain. For instance the *SHIFT*, *MODE* and *OFF* function keys arrangement is seen in 92% of the top best layouts. The same statistic is obtained for the *DEL*, *STO*, *M+*, *RCL* and *AC* function keys i.e. 92% of all evolved solutions, indicated a possible natural arrangement for those components. The ANS key is found bottom right of the right area in 60% of the cases indicating yet again a possibility of an appropriate location for this component.

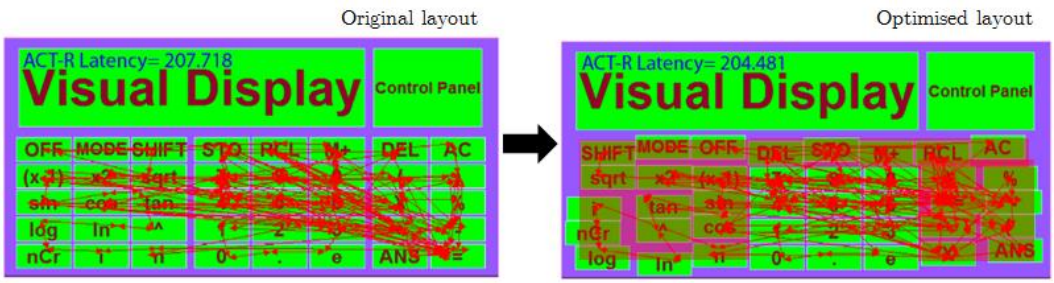



Figure 89: optimised user interaction (4 areas) 10000 generations

The user interaction on both original and optimised interface for the *four areas* design approach can be seen in Figure 89. The flow of the actions between the keys on the optimised layout is more systematic, minimising the distance between the key that are sequentially used in addition to minimise the cognitive load whilst exercising the user interface. Appendix K includes all the top 5% layouts in terms of ACT-R timing and provides more evidence that layouts optimised by TOISE based from the original layout shown is Figure 88 are very similar and provide enough information to designer to place the components in better locations on the interface.

The optimisation process when dealing with the three designated areas i.e. *three areas* design method provides better cognitive latencies overall mostly due to the increased freedom the genetic algorithm has in placing the components on the interface when evolving the layout solutions. The best top 5% is shown in Figure 90.

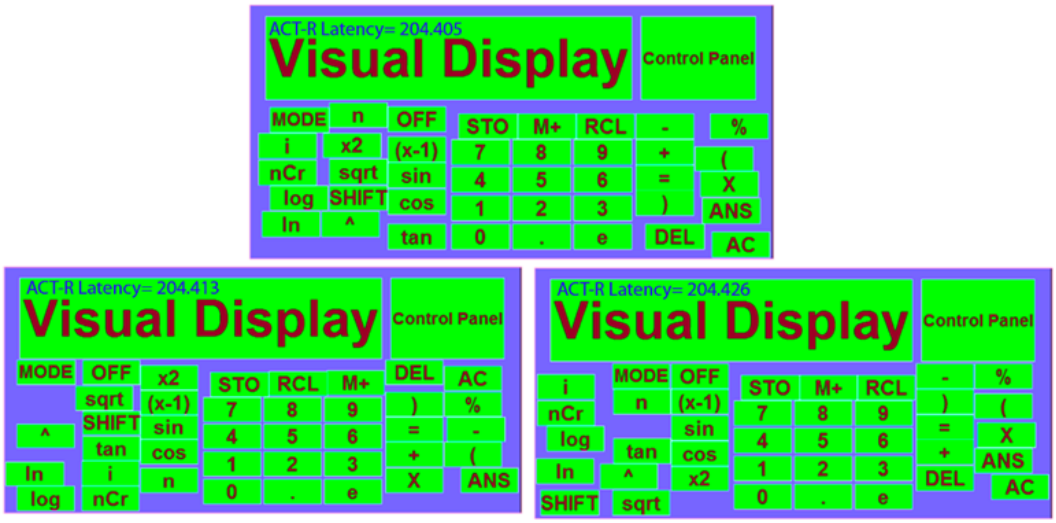



Figure 90: Top three optimised (< 2* STDEV) layouts
using (3 areas) 10000 generations

However, the layouts are not as well laid as they were with the layouts designed with *four areas* approach due to the lack of constraints. However, they provide useful information regarding ways to optimise the calculator interface despite this fact. For instance the *DEL*, *ANS* and *AC* function keys arrangement is found in 60% of the top layouts. See Appendix L to peruse the other top layouts. The – and *%* keys arrangement is present in 50% of the best layouts. The arrangement for the *STO*, *RCL* and *M+* contains only two possible layouts both of which are displayed in Figure 90, each one 50% of the top best layouts. Any other layouts for those keys were not evolved by TOISE. Moreover, the *SIN* and *COS* function keys are located in the same locations in all but one solutions, which may indicate to the interface designer a natural position for those two keys.

Other results were obtained by fluctuating the number of generations of the genetic algorithms and experiments were carried out with 5000 and 20000 generations to investigate the quality of the solutions found by TOISE in terms of gain / loss related to ACT-R timing. Figure 91 displays the ACT-R latencies distribution curves for those experiments.

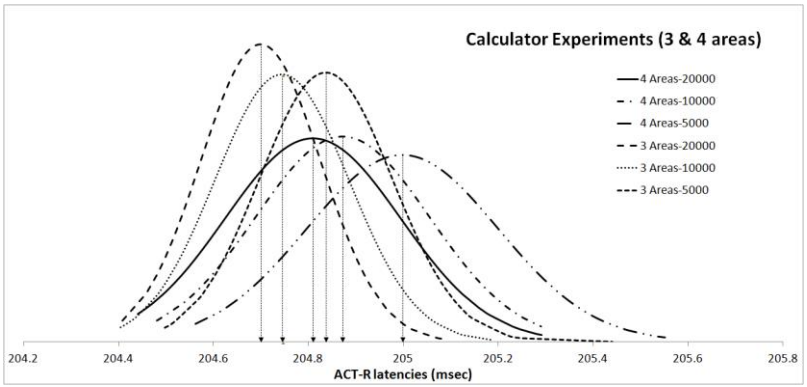



Figure 91: Comparison of 3/4 areas experiments
using different generation number

Perusing at Table 21, one realises that there is significant gains in increasing the generation size from 5000 to 20000 as better average ACT-R latencies are found with both the *three areas* and *four areas* methods. The latencies decrease from 204.836, 204.745 down to 204.699 for 5000, 10000 and 20000 generations respectively using the *three areas* method, and from 204.998, 204.872 down to 204.809 for the same generations using the *four areas* approach. The overall better performance for the *three areas* method over the *four areas* approach is expected as the gain of freedom of placement of the components helps the GA to explore the search space more thoroughly hence the increase in performance for this method.

Table 21: Comparing different approaches using different generation size for GA

**Multiple Comparisons of 3 and 4 areas with different GA generation size**

| | KIND | Compared to | Mean Difference (I-J) | Std. Error | Sig. | 95% Confidence Interval | |
|---|---|---|---|---|---|---|---|
| | | | | | | Lower Bound | Upper Bound |
| Tukey HSD | 3 areas - 10000 gens | 3 areas - 20000 gens | .0456200* | .0147799 | .025 | .003447 | .087793 |
| | | 3 areas - 5000 gens | -.0915680* | .0147799 | .000 | -.133741 | -.049395 |
| | | 4 areas - 10000 gens | -.1272000* | .0147799 | .000 | -.169373 | -.085027 |
| | | 4 areas - 20000 gens | -.0646360* | .0147799 | .000 | -.106809 | -.022463 |
| | | 4 areas - 5000 gens | -.2534600* | .0147799 | .000 | -.295633 | -.211287 |
| | 3 areas - 20000 gens | 3 areas - 10000 gens | -.0456200* | .0147799 | .025 | -.087793 | -.003447 |
| | | 3 areas - 5000 gens | -.1371880* | .0147799 | .000 | -.179361 | -.095015 |
| | | 4 areas - 10000 gens | -.1728200* | .0147799 | .000 | -.214993 | -.130647 |
| | | 4 areas - 20000 gens | -.1102560* | .0147799 | .000 | -.152429 | -.068083 |
| | | 4 areas - 5000 gens | -.2990800* | .0147799 | .000 | -.341253 | -.256907 |
| | 3 areas - 5000 gens | 3 areas - 10000 gens | .0915680* | .0147799 | .000 | .049395 | .133741 |
| | | 3 areas - 20000 gens | .1371880* | .0147799 | .000 | .095015 | .179361 |
| | | 4 areas - 10000 gens | -.0356320 | .0147799 | .153 | -.077805 | .006541 |
| | | 4 areas - 20000 gens | .0269320 | .0147799 | .452 | -.015241 | .069105 |
| | | 4 areas - 5000 gens | -.1618920* | .0147799 | .000 | -.204065 | -.119719 |
| | 4 areas - 10000 gens | 3 areas - 10000 gens | .1272000* | .0147799 | .000 | .085027 | .169373 |
| | | 3 areas - 20000 gens | .1728200* | .0147799 | .000 | .130647 | .214993 |
| | | 3 areas - 5000 gens | .0356320 | .0147799 | .153 | -.006541 | .077805 |
| | | 4 areas - 20000 gens | .0625640* | .0147799 | .000 | .020391 | .104737 |
| | | 4 areas - 5000 gens | -.1262600* | .0147799 | .000 | -.168433 | -.084087 |
| | 4 areas - 20000 gens | 3 areas - 10000 gens | .0646360* | .0147799 | .000 | .022463 | .106809 |
| | | 3 areas - 20000 gens | .1102560* | .0147799 | .000 | .068083 | .152429 |
| | | 3 areas - 5000 gens | -.0269320 | .0147799 | .452 | -.069105 | .015241 |
| | | 4 areas - 10000 gens | -.0625640* | .0147799 | .000 | -.104737 | -.020391 |
| | | 4 areas - 5000 gens | -.1888240* | .0147799 | .000 | -.230997 | -.146651 |
| | 4 areas - 5000 gens | 3 areas - 10000 gens | .2534600* | .0147799 | .000 | .211287 | .295633 |
| | | 3 areas - 20000 gens | .2990800* | .0147799 | .000 | .256907 | .341253 |
| | | 3 areas - 5000 gens | .1618920* | .0147799 | .000 | .119719 | .204065 |
| | | 4 areas - 10000 gens | .1262600* | .0147799 | .000 | .084087 | .168433 |
| | | 4 areas - 20000 gens | .1888240* | .0147799 | .000 | .146651 | .230997 |

**Descriptive Statistics**

| | N | Minimum | Maximum | Mean | | Std. Deviation | Variance |
|---|---|---|---|---|---|---|---|
| | Statistic | Statistic | Statistic | Statistic | Std. Error | Statistic | Statistic |
| **3 areas - 5000 gens** | 250 | 204.4970 | 205.4400 | 204.836448 | .0088412 | .1397920 | .020 |
| **4 areas - 5000 gens** | 250 | 204.5620 | 205.5520 | 204.998340 | .0127419 | .2014672 | .041 |
| **3 areas - 10000 gens** | 250 | 204.4050 | 205.1860 | 204.744880 | .0089147 | .1409532 | .020 |
| **4 areas - 10000 gens** | 250 | 204.4810 | 205.2900 | 204.872080 | .0116010 | .1834286 | .034 |
| **3 areas - 20000 gens** | 250 | 204.4080 | 205.0910 | 204.699260 | .0080072 | .1266048 | .016 |
| **4 areas - 20000 gens** | 250 | 204.4410 | 205.2910 | 204.809516 | .0116893 | .1848233 | .034 |

It is worth mentioning that the *four areas* running with 10000 and 20000 generations is not significantly better in terms of the ACT-R timings as the *three areas* methods which means that the latter running at 5000 generations i.e. very fast, seems to outperform most of the *four areas* methods, yet again demonstrating that constraining an interface design with areas, can impede on the quality of usability performance, as a trade-offs with design cohesion, style and heuristic principles.

Table 22 provides statistical evidence that the *three areas* approach significantly benefit of longer runs with means decreasing from 204.836 down to 204.699 as the length of the runs increases, a trend that is similar for the *four areas* approach tabulated in table 23 for which the ACT-R latencies decrease from 204.998 down to 204.809.

Table 22: Comparing different generation size for 3 areas approach

| Algorithm: 3 areas | N | Minimum | Maximum | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|---|
| [1]: 5000 generations | 250 | 204.4970 | 205.4400 | 204.836448 | .1397920 | .0088412 |
| [2]: 10000 generations | 250 | 204.4050 | 205.1860 | 204.744880 | .1409532 | .0089147 |
| [3]: 20000 generations | 250 | 204.4080 | 205.0910 | 204.699260 | .1266048 | .0080072 |

| T-Test | | | | | |
|---|---|---|---|---|---|
| Test type | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference |
| [1] & [2] | 7.293 | 498 | .000 | .0915680 | .0125554 |
| [1] & [3] | 11.501 | 498 | .000 | .1371880 | .0119282 |
| [2] & [3] | 3.807 | 498 | .000 | .0456200 | .0119827 |

Comparing the two approaches together in terms of cognitive and motor function load, it is obvious that the gains are very small with a percentage decrease of 0.067% for the *three areas* and 0.092% for the *four areas* method.

Table 23: Comparing different generation size for 4 areas approach

| Algorithm: 4 areas | N | Minimum | Maximum | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|---|
| [1]: 5000 generations | 250 | 204.5620 | 205.5520 | 204.998340 | .2014672 | .0127419 |
| [2]: 10000 generations | 250 | 204.4810 | 205.2900 | 204.872080 | .1834286 | .0116010 |
| [3]: 20000 generations | 250 | 204.4410 | 205.2910 | 204.809516 | .1848233 | .0116893 |

| T-Test | | | | | |
|---|---|---|---|---|---|
| Test type | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference |
| [1] & [2] | 7.327 | 498 | .000 | .1262600 | .0172320 |
| [1] & [3] | 10.920 | 498 | .000 | .1888240 | .0172915 |
| [2] & [3] | 3.799 | 498 | .000 | .0625640 | .0164688 |

In terms of layout quality, Appendix11 and Appendix L display the top best 5% of evolved designs. Table 24 and Table 25 tabulate the most prominent similar key arrangements for both experiments, providing a similarity index for each approach. The overall similarity index formula is shown below:

$$S = \frac{\sum_{j=1}^{n} r_j}{n} \qquad (6.1)$$

Where S is the similarity index (percentage) for a series of layouts, n is the number of layouts under scrutiny and $r$ is the number of re-occurrence as a percentage of a certain amount of key or key arrangements that appears in all the layouts.

The index for the *three areas* approach is 65.71% and 80.86% for the *four areas* method. This tells that using the latter design method provides more consistent layout arrangements over a number of runs as it would by using the former design approach. Hence even though the scoring in terms of usability is lower when using this method, the

compensation is the provision of better, cohesive and similar designs over a number of TOISE runs.

The same experiment was carried out (250 runs) with only 5000 generations using both approaches. Figure 92 displays the top best layout found in those runs for the *three areas* design. What is noticeable is the user interaction compared to the original design. Perusing the top 5% design layout in Appendix P a certain amount of similarity between the solutions is perceived. For instance the *[STO, RCL, M+]* function key location are similar in 83% of these top scorers. Table 24 provides a listing with the most visible component placement similarities amongst the optimised designs. This method scored 66.20% and is the lowest of all the methods.

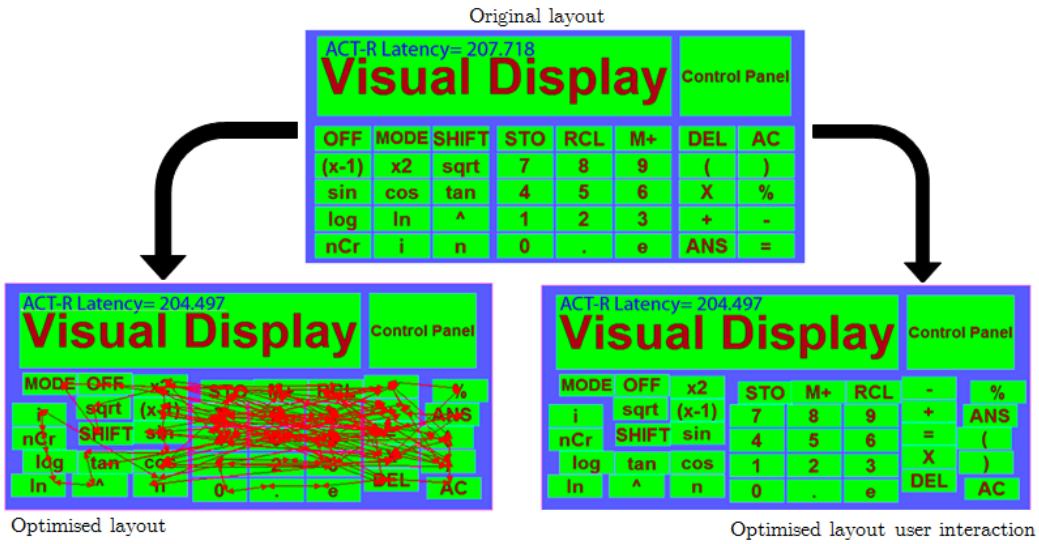



Figure 92: Top optimised (< 2* STDEV) layout
using (3 areas) 5000 generations

Many key arrangements can be seen with 5000 generations used alongside the *three areas* design and some consistency in the design exists.

Using a *four areas* approach with only 5000 generations exhibits the same type of behaviour as the three areas design method See Figure 93. For instance the *[DEL, STO, M+] and [RCL, AC]* arrangement occurs in all the optimised layouts. The [*SIN]* function is similar. The top 5% best optimised designs are displayed in Appendix N.

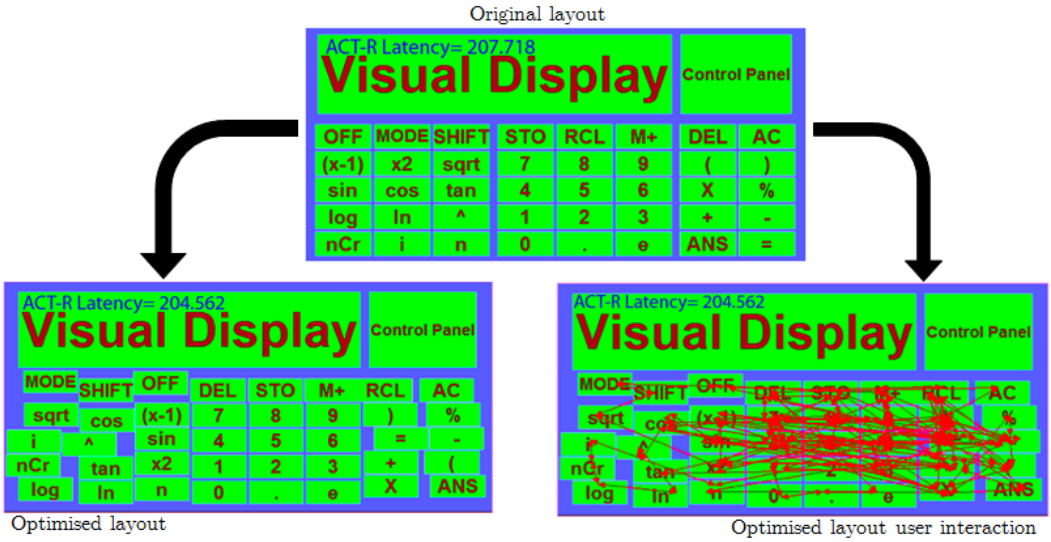



Figure 93: Top optimised (< 2* STDEV) layout
using (4 areas) 5000 generations

A look at Table 25 shows that the score for this design approach with 5000 generations is 78.25% which is much higher than the *three areas* design approach indicating a higher level of cohesion and consistency in terms of solutions throughout the design solutions.

The amount of generations for the genetic algorithm was increased to 20000 generations for both design methods. Figure 94 shows the best layout found over 250 runs for the *three areas* approach.

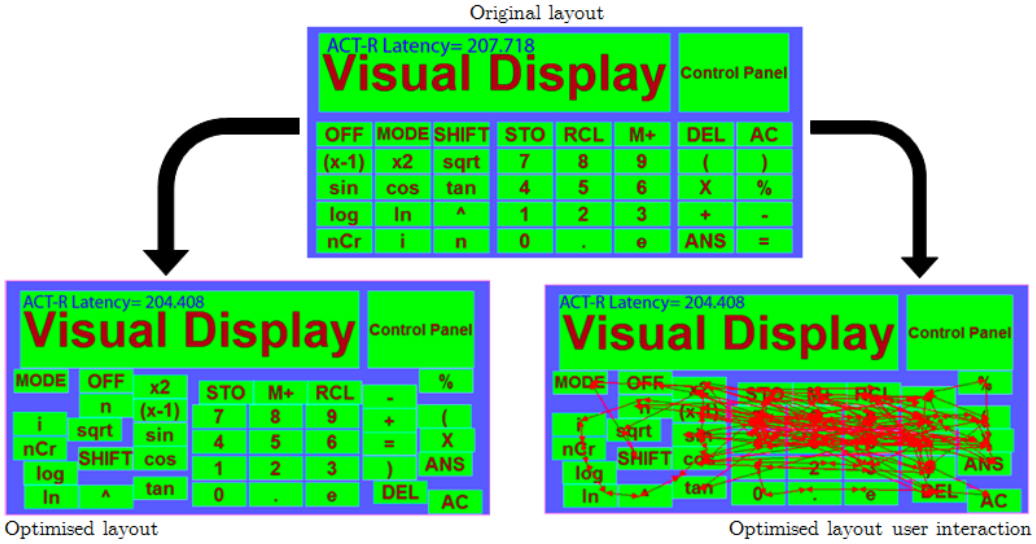



Figure 94 Top optimised (< 2* STDEV) using *three areas*
with 20000 GA generation size

Perusing at Appendix M and Figure 94 top 5% layout arrangements, one can see that many layouts are consistent in terms of layout placement. For instance the *[MODE, OFF, X2]* component arrangement is seen over 61%, the *[COS]* and *[%]* functions keys are

placed similarly in 69% of the cases. In Table 24 which provides more examples has a similarity index for the main similar key arrangements of 60.56% for this experiment.

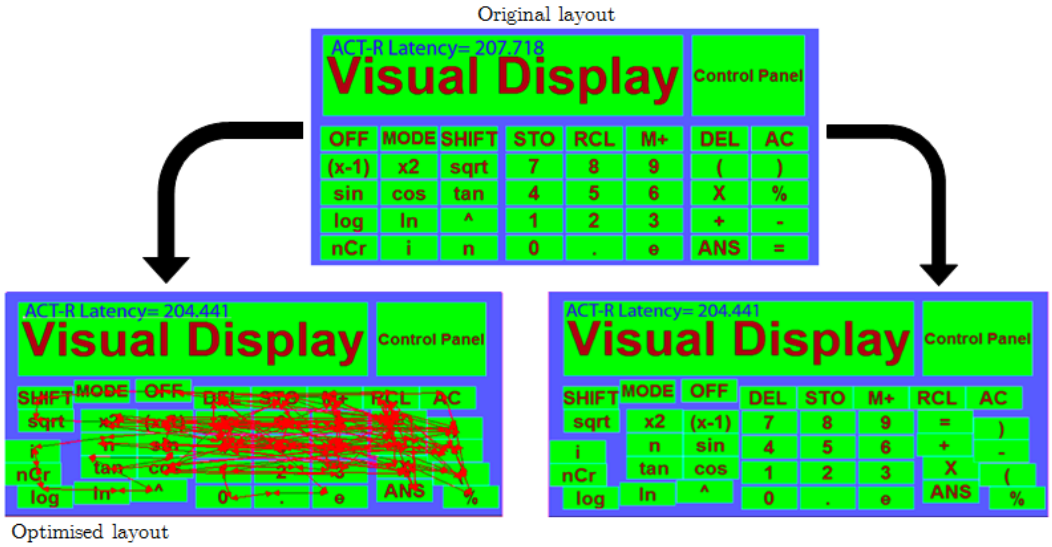



Figure 95: Top optimised (< 2* STDEV) layout
using (4 areas) 20000 generations

Table 24: Similarity of component placement (3 areas)

| Type | Function key | Percentage similarity between designs | Similarity |
|---|---|---|---|
| **3 areas 5000 gens** | MODE,RCL,X2 | 67% | |
| | STO,RCL,M+ | 83% | |
| | (X-1) | 75% | |
| | nCr | 42% | |
| | COS | 75% | |
| | SIN | 95% | 66.20% |
| | n | 50% | |
| | % | 75% | |
| | DEL,ANS,AC | 50% | |
| | - | 50% | |
| 3 areas 10000 gens | X2 | 60% | |
| | Mode | 70% | |
| | (x-1) | 90% | |
| | SIN | 100% | |
| | COS | 80% | |
| | LOG | 50% | |
| | Ln | 70% | 65.71% |
| | STO,M+,RCL | 50% | |
| | STO,RCL,M+ | 50% | |
| | DEL,ANS,AC | 50% | |
| | % | 60% | |
| | = | 70% | |
| | - | 60% | |
| **Type** | **Function key** | **Percentage similarity between designs** | **Similarity** |
| | OFF | 60% | |
| 3 areas 20000 gens | MODE,OFF,X2 | 61% | |
| | STO,M+,RCL | 54% | |
| | STO,RCL,M+ | 46% | |
| | COS | 69% | |
| | SIN | 77% | 60.56% |
| | LOG | 62% | |
| | % | 69% | |
| | DEL,ANS,AC | 69% | |
| | X | 38% | |

The same experiment using the four areas approach was carried over 250 runs and the top solution is displayed in Figure 95. The best top 5% are displayed in Appendix O. Perusing these layouts, one can recognise a lot of cohesion and similarity amongst the range of optimised designs. Table 25 has a similarity score of 81.73%, showing that key arrangements such as *[SHIFT, MODE, OFF]*, *[DEL, STO, M]*, *[RCL, AC]* and many others occur over 100% of the cases. This is the top scorer over the entire set of experiments with regards to similarity of layout evolution over 250 runs.

Table 25: Similarity of component placement (4 areas)

| Type | Function key | Percentage similarity between designs | Similarity |
|---|---|---|---|
| **4 areas 5000 gens** | SHIFT,MODE,OFF | 80% | 78.25% |
| | SQRT | 58% | |
| | (X-1) | 92% | |
| | n | 67% | |
| | SIN | 100% | |
| | COS | 58% | |
| | DEL,STO,M+ | 100% | |
| | RCL,AC | 100% | |
| | ANS | 67% | |
| | = | 58% | |
| | - | 67% | |
| | X | 92% | |
| **4 areas 10000 gens** | (X-1) | 100% | 80.86% |
| | SQRT | 83% | |
| | X2 | 67% | |
| | I | 83% | |
| | COS | 92% | |
| | LOG | 83% | |
| | LN | 75% | |
| | NCr | 83% | |
| | SIN | 92% | |
| | N | 58% | |
| | DEL,STO,M+,RCL,AC | 100% | |
| | X | 100% | |
| | ANS | 58% | |
| | % | 58% | |
| **4 areas 20000 gens** | SHIT,MODE,OFF | 100% | 81.73% |
| | DEL,STO,M+ | 100% | |
| | RCL,AC | 100% | |
| | SQRT | 75% | |
| | (X-1) | 100% | |
| | N | 83% | |
| | SIN | 100% | |
| | COS | 92% | |
| | X | 92% | |
| | TAN | 67% | |
| | LOG | 50% | |
| | = | 67% | |
| | % | 67% | |
| | X | 83% | |
| | - | 50% | |

### 6.1.5.2 Discussion

The results discussed above are very encouraging. TOISE managed to find layout arrangements that minimise the cognitive and motor functions load for all of its solutions that were found during all the experiments conducted i.e. all the ACT-R timing were much below the original latency of 207.718 seconds.

The experiments conducted and discussed in this chapter are complex as the genetic algorithm had to deal with a very restricted space where to position the components, with static components that provide barriers to the free movements. Thus a learning process took place which helped to find good locations that do not resulted in penalties taking into account the ACT-R pressure into account first, which was then followed by another learning phase that focused mainly on the ACT-R pressure to fine-tune the layout and optimise the interface layout evolving an interface that was better fitted for human interaction.

In these experiments, some issues arose. First, varying some of the genetic algorithm parameters can lead to improvements and drive the GA to evolve better solutions. However the cost for these small gains in performance comes with a great increase of extra computations and run time taken to evolve those new optimised layouts. For instance a gain of 0.062% (from 204.99834 down to 204.87208) when doubling the number of generations of the system from 5000 to 1000 generations, may not be considered as valuable (even though 120.626 milliseconds is significant), compared to a gain 1.327% (from 207.718 down to 204.99834 i.e. 2.71966 seconds) when optimising the original layout using only 5000 generations, which is in terms of human cognition a very significant value. However this lack of noticeable gain is compensated as when the number of constraints such as areas is integrated into the interface design, and this is combined with an increase of generations i.e. runs lasting longer, the design layouts are more consistent, cohesive and despite the fact that may not be the best solution with regards to ACT-R latencies and human interaction, they provide still a major gain in usability compared to the original design.

TOISE dealt with this increase of complexity and generated consistent and cohesive layouts, and highlighted many areas that can be beneficial in terms of design i.e. providing ideas about the most likely locations for interface components etc. It is believed that TOISE can provide assistance to any interface designers with regards to usability performance.

In terms of comparison with a local search algorithm, the main advantage is the speed at which the GA finds the best solutions as the LS approach does seem to be able to find good solutions provided very long run times. As suggested in all experiments, the difference in computation times between the GA and LS approaches is in the order of hours for the GA and days for the LS method.

# Chapter 7. Conclusion

## 7.1 Development and issues

The ACT-R cognitive architecture is a mature computer-based simulation framework that embodies theories of human cognition, human perception and motor functions. It provides a formidable base for modelling simulated users and has been used to develop simulated users in many projects. However, developing models for ACT-R is non-trivial task, which demands expertise not only about the ACT-R framework itself but also about the embedded theories of human cognition, human vision and motor functions, in addition to a knowledge of the LISP programming language to extend and embed those models with functionality. It is needless to mention that developing cognitive models is time consuming, demanding expertise that often remains in the realm of specialists and is often an expensive process and to be honest most probably far too complicated for most software designers, UI designers or cognitive scientists.

The main usability assessment as an interactive process in software development is often undermined because of cost both in terms of expenditure and time. Its reliance of potential users and the organisational logistics makes this process difficult to undertake as an iterative process. The response to this state of affair by the HCI community was the development of specialised tools. We have discussed in this thesis a system called CogTool, which has the potential to render usability testing more approachable to user interface designers that are not usability specialists, and has had an impact in many software developments by simulating users using ACT-R models. However this type of software is only assessing already developed user interfaces and does not present ideas of optimised interface layout.

The concept behind the system presented in this thesis called TOISE attempts to bridge this gap by using simulated users, based on ACT-R models within an evolutionary framework to proposed optimised user interfaces based on the original design idea.

In this respect, TOISE succeeded in its primary proposal as TOISE not only provides a way to exercise user interaction on a user interface using a fully functional software using its ROBOT module but it proposes evolved optimised interface layouts.

However, this research has stretched the limits of feasibility when interfacing the ACT-R framework with external processes both in terms of functionality and speed of execution and trade-offs were necessary to complete the software prototype to include the genetic

algorithm with its visual interface to run at a sensible speed that allow it to be used as a toolkit.

There is much in the literature about the use of surrogate fitness evaluations when dealing with evolutionary algorithms. It becomes often an option when the fitness assessment of the individuals in a population is highly time-consuming, computationally resource intensive or complex. In those instances surrogate fitness approximation is often the only practical direction available. This domain is the domain of surrogate-assisted evolutionary computation. (Zhou et al. 2007; Jin 2011)

Bearing this in mind, it is worth noting that the first version of TOISE was created to interact directly with the underlying ACT-R framework, where a set of small composite primitive actions for a user interaction were modelled and ran to provide the overall ACT-R latency. As explained in earlier sections, the nature of the Lisp environment does not lend itself to easy interfacing. Furthermore, the fitness assessments necessary for each of the population individuals of the optimising algorithm were time-consuming, computationally intensive and complex, leading to extremely long run times. Moreover due to the intensity of resource demands and some spurious memory leaks (as discussed in Section 5.11), the addition of further functionality became questionable with this version, a state that was not factorised at the beginning of the project. This provided the pretext and the drive for the creation of the next version of TOISE with the removal of the ACT-R architecture which led to the porting and re-coding of the relevant ACT-R functionality into C++ native code. As mentioned in section 5.12 which refers also to the source code listings in Appendix J, the porting of ACT-R functionality into native code was only made possible because of the atomic nature of the actions that are modelled; a granularity that made this task possible and allowed for the development of extra functionality such as more powerful mutation operators all of which leading to a noticeable performance boost of the TOISE system.

Another factor that hampered the completion of the project and introduced some limitations of functionality was the modelling of interaction and user interaction recording into virtual models within the main software. Porting an entire Java API dealing with java interface to a few C++ classes within the project proved challenging and not as straightforward as originally thought, and led to some limitations in terms of interfaces that can be properly optimised. For instance some of the Layout Managers (i.e. Grid Bag Layout) are not consistent with TOISE as their intricate ways of placing the components in the interface was too intricate to interpret given the time frame of the TOISE development.

In terms of usability, one of the main focuses that were perceived from the start of the TOISE design was the removal of all technical settings with regards to the genetic algorithm so to address one of the major concerns of adoption of the toolkit by software designers. In this respect, many attempts have been made to alleviate the amount of required actions (i.e. automatic mutation and crossover rates etc.) and thus TOISE succeeded in this respect, but there is still a small amount of general parameters, most of them often left at their default values, that affect the TOISE system performance when dealing with intricate interfaces or complex interaction i.e. penalties pressure, population size and residual diversity. However as seen in the previous chapter, some of those settings are fine-tuning parameters that can be left at their default values for non-technical minded users.

A point that has to be noted was the difficulties of creating mutation operators that suited this type of project this thesis presents i.e. unsystematic heuristics. Finding ways to move components on an interface that helps the GA evolving solutions and not impeding it was in itself a challenging task.

Finally, there is another aspect that should be mentioned in this discussion. The scope of this research was mainly focussing on creating and using surrogate users to assess the quality and usability of user interfaces given some human recorded interaction. However in terms of design there are other heuristic algorithms that when applied to user interfaces can help improve usability, namely the concerns of aesthetic and graphical design, which is often omitted by HCI researchers. Lack of aesthetic can lead to as much user frustration as does poorly designed interfaces in terms of cognitive load for instance. Ling for instance (Ling et al. 2000), proposes a set of measures that deals with user perception and appreciation of UI focusing on balance (weighting of the interface components on either sides of the horizontal and vertical axis), equilibrium (configuration of the interface objects to implement a centre of forces which should coincide with the centre of the interface, often used for many art work visual composition ), symmetry (balanced distribution of equivalent interface objects about a common vertical and horizontal line, as symmetry creates balanced states), sequence (arrangements of interface components that facilitate human eye movements by minimizing visual scanning in configuring the interface layout so that users can move thought the layout from top-left to bottom-right of the layout by attending the least number of visual targets, leading therefore to faster performance) and complexity (measure of layout order and complexity based on Birkhoff 1923) (Ling et al. 2000; Balinsky 2006). These measures could therefore in theory be used by TOISE as extra objectives during the assessment of layout

fitness by the optimiser to increase TOISE capability not only to deliver designs that are usable in terms of human cognition and motor functions but also usable in terms of proportions and aesthetic, the latter so important in today's new technologies.

A study carried out by Abubakar (Abubakar 2010), focussed on providing a CogTool and user evaluation on the findings published by Golovine & et al. 2010 regarding optimisation of the ubiquitous Bell's keypad phone layout. This research provided qualitative and quantitative measures of evaluation comparing completion tasks using the original Bell's keypad alongside two optimised layout obtained by TOISE, using both CogTool and a limited amount of real users (ten participants) which used a specialised solution tester that recorded completion tasks to assess completion times across the three different layouts. In addition a questionnaire was created to provide more insight with regards to some of the reasons that led to some of the finding obtained during the course of this study. On one hand, the findings showed that the TOISE layouts performed better than the Bell's layout when tasks were modelled in CogTool. On the other hand, the user evaluation came to a different conclusion whereby a significant amount of users preferred the ubiquitous Bell's layout rather than the TOISE optimised interfaces. Despite the fact that the user performance results showed a significant variance between participants; a point referred to in the literature on many occasions (Ritter et al. 2002), these findings were expected, a fact reinforced by the qualitative survey where 37% of the participants mentioned their familiarity with this keypad and the years of practice as the main factor for their high performance with the original layout. On a positive note, these findings also suggested that with practice, the completion timing for all three layouts decreased which yet again agrees with the opinion that CogTool (and therefore TOISE's predictions) simulates expert users rather than novices.

Given the results provided by this study, a major improvement would be TOISE to simulate novice users. One possible way to provide this functionality would be to have the models actively searching the interface for the numbers used rather than suggesting the right location for the objects as it is currently done. This points us to the work carried by Halbrügge (Halbrügge et al. 2007) on ACT-CV. However given the complexity of the ACT-R models that would be needed in TOISE, this would imply porting much of the ACT-R framework to native code, which in itself would be a major task in itself.

## 7.2 Contributions and achievements

TOISE offers possibilities in the field of user interface design not only to assess user interface but to optimise their layout to enhance their usability using a Darwinian approach to evolution in conjunction to simulated users; a approach named in this thesis as EvoCog.

1. TOISE provides a user interaction recorder that can record user interaction on fully functional Java Software.

2. TOISE provides user interaction and user interface models as XML models that can be easily re-scripted to suit various flavours of development.

3. TOISE optimises a user interface both in terms of human cognitive and motor functions load using an evolutionally approach that is fast and reliable, providing coherent and consistent optimised user interface designs over a number of runs.

## 7.3 Future work

The discussion in this thesis demonstrates that the concept embedded in TOISE is potentially a valuable contribution to the HCI community. However TOISE needs to be re-designed from ground up and to be modularised to provide a useful tool.

Moreover, as stated in the last section, there are many settings that should be removed and automated. The concept of pre-run should therefore be extended to more areas to increase its usability. As it stands, a certain amount of knowledge is required in the field of evolutionary algorithms to understand the effect and causes that affect its functionality.

Usability heuristics have long been used in design and aesthetics plays a significant role in the overall value of an interface design. Ling and colleagues have proposed a set of metrics to measure the balance, equilibrium, symmetry and sequence of screen and interface design (Ling et al. 2000).

The TOISE approach is capable of taking into account those criteria when assessing interface designs which would be a valuable addition to the system.

# Appendices

## APPENDIX A: A TYPICAL G2A TRANSLATION USING A GOMSL MODEL

```
Define_model: "JCG Dragging Example"
Starting_goal is DragDrop1 Action.

Visual_object: SW1_Object
  Content is "Object1".

Visual_object: SW2_Object
   Content is "Object2".

Method_for_goal: DragDrop1 Action
    Step 2. Look_for_object_whose Content is "Object1" and_store_under <src-target>.
    Step 3. Point_to <src-target>; Delete <src-target>.
    Step 4. Hold_down mouse_button.
    Step 5. Look_for_object_whose Content is "Object2" and_store_under <dest-target>.
    Step 6. Point_to <dest-target>; Delete <dest-target>.
    Step 7. Release mouse_button.
```

Content of the drag-drop-action-2.goms model file

**This model need to be processed using after ensuring that the current package is indeed G2A running:**

*(in-package G2A)*

**Then running:**

*(input-model (pathname "G2A:data;jcg-drag-model.goms"))*

**Once the model is created in the running ACT-R, it must be saved to file:**

*(output-model *current-model* (pathname "G2A:data; GOMS-ACTR-model.lisp"))*

```
;;;*****************************************************************
;;; Preamble
(CLEAR-ALL) (PM-RESET) (PM-START-HAND-AT-MOUSE)
;;;*****************************************************************
;;; Chunk types
(CHUNK-TYPE VISUAL-ITEM %ID %SCREEN-XY %SCREEN-X %SCREEN-Y CONTENT)
(CHUNK-TYPE AUDITORY-ENCODING %ID)
(CHUNK-TYPE LTM-ITEM %ID)
(CHUNK-TYPE TASK-ITEM %ID)
;;;*****************************************************************
;;; Chunks
(ADD-DM
 (VISUAL-SW1-OBJECT ISA VISUAL-LOCATION KIND "object1" SCREEN-X 809 SCREEN-Y 144)
 (SW1-OBJECT ISA VISUAL-ITEM %ID VISUAL-ITEM1 %SCREEN-XY EMPTY   %SCREEN-X EMPTY %SCREEN-Y EMPTY CONTENT "object1")
 (VISUAL-SW2-OBJECT ISA VISUAL-LOCATION KIND "object2" SCREEN-X 354   SCREEN-Y 163)
 (SW2-OBJECT ISA VISUAL-ITEM %ID VISUAL-ITEM2 %SCREEN-XY EMPTY   %SCREEN-X EMPTY %SCREEN-Y EMPTY CONTENT "object2"))
;;;*****************************************************************
;;; Goal type
(CHUNK-TYPE GOAL %STATE %LAST-LOCATION %NEXT-LOCATION %RETURN  %PREVIOUS-RETURN %R0 %R1 %R2 %R3 %R4 [TARGET])
(CHUNK-TYPE RECORD %RETURN %PREVIOUS-RETURN [TARGET])
;;;*****************************************************************
;;; Top-level goal
(ADD-DM
 (GOAL ISA GOAL %STATE DRAGDROP1-ACTION-0 [TARGET] EMPTY %LAST-LOCATION
  EMPTY %NEXT-LOCATION EMPTY %RETURN EMPTY %PREVIOUS-RETURN EMPTY %R0
  EMPTY %R1 EMPTY %R2 EMPTY %R3 EMPTY %R4 EMPTY))
;;;*****************************************************************
;;; DRAGDROP1-ACTION
(P FIXED-Think-of-How-to-do-it-best-32
  =GOAL>
    ISA             GOAL
    %STATE          DRAGDROP1-ACTION-0
==>
  =GOAL>
    %STATE          DRAGDROP1-ACTION-1
  !OUTPUT! (FIXED-THINK-OF-HOW-TO-DO-IT-BEST-32)
  !OUTPUT! (IN 'DRAGDROP1-ACTION)
  !EVAL! (START-TIMING 'DRAGDROP1-ACTION
               (- (PM-TIME)
                 (PRODUCTION-DURATION '|FIXED-Think-of-How-to-do-it-best-32|)))
)
(P T38
  =GOAL>
    ISA             GOAL
    %STATE          DRAGDROP1-ACTION-3
  =VISUAL>
    ISA             TEXT
  - VALUE             "object1"
  =VISUAL-LOCATION>
    ISA             VISUAL-LOCATION
==>
  =GOAL>
    %STATE          DRAGDROP1-ACTION-1
  !OUTPUT! ("Escape: object1 not found")
)
(P FIND-34
  =GOAL>
    ISA             GOAL
    %STATE          DRAGDROP1-ACTION-1
  =VISUAL-STATE>
    ISA             MODULE-STATE
    MODALITY          FREE
==>
  =GOAL>
    %STATE          DRAGDROP1-ACTION-2
  +VISUAL-LOCATION>
    ISA             VISUAL-LOCATION
```

```
)
(P ATTEND-35
  =GOAL>
     ISA             GOAL
     %STATE            DRAGDROP1-ACTION-2
  =VISUAL-LOCATION>
     ISA             VISUAL-LOCATION
==>
  =GOAL>
     %STATE            DRAGDROP1-ACTION-3
  +VISUAL>
     ISA             VISUAL-OBJECT
     SCREEN-POS          =VISUAL-LOCATION
)
(P ENCODE-36
  =GOAL>
     ISA             GOAL
     %STATE            DRAGDROP1-ACTION-3
  =VISUAL>
     ISA             TEXT
     VALUE             =TEXT
     VALUE             "object1"
  =VISUAL-LOCATION>
     ISA             VISUAL-LOCATION
==>
  =GOAL>
     [TARGET]          =VISUAL-LOCATION
     %STATE            DRAGDROP1-ACTION-4
  !OUTPUT! ("Look-for-object-whose ((IS CONTENT Object1))")
)
(P POINT-TO-39
  =GOAL>
     ISA             GOAL
     %STATE            DRAGDROP1-ACTION-4
  =MANUAL-STATE>
     ISA             MODULE-STATE
     MODALITY          FREE
==>
  =GOAL>
     %STATE            DRAGDROP1-ACTION-5
  +MANUAL>
     ISA             HAND-TO-MOUSE
  !OUTPUT! ("Home-to")
)
(P T40
  =GOAL>
     ISA             GOAL
     [TARGET]          =TARGET41
     %STATE            DRAGDROP1-ACTION-5
  =MANUAL-STATE>
     ISA             MODULE-STATE
     MODALITY          FREE
==>
  =GOAL>
     %LAST-LOCATION       =TARGET41
     %STATE            DRAGDROP1-ACTION-6
  +MANUAL>
     ISA             MOVE-CURSOR
     LOC              =TARGET41
  !OUTPUT! ("Point-to ~A" =TARGET41)
)
(P HOLD-DOWN-42
  =GOAL>
     ISA             GOAL
     %STATE            DRAGDROP1-ACTION-6
  =MANUAL-STATE>
     ISA             MODULE-STATE
     MODALITY          FREE
==>
  =GOAL>
     %STATE            DRAGDROP1-ACTION-7
  +MANUAL>
     ISA             HAND-TO-MOUSE
  !OUTPUT! ("Home-to")
)
(P T43
  =GOAL>
     ISA             GOAL
     %STATE            DRAGDROP1-ACTION-7
  =MANUAL-STATE>
     ISA             MODULE-STATE
     MODALITY          FREE
==>
  =GOAL>
     %STATE            DRAGDROP1-ACTION-8
  +MANUAL>
     ISA             PRESS-MOUSE
)
(P T49
  =GOAL>
     ISA             GOAL
     %STATE            DRAGDROP1-ACTION-10
  =VISUAL>
     ISA             TEXT
  - VALUE             "object2"
  =VISUAL-LOCATION>
     ISA             VISUAL-LOCATION
==>
  =GOAL>
```

```
    %STATE              DRAGDROP1-ACTION-8
  !OUTPUT! ("Escape: object2 not found")
)
(P FIND-45
  =GOAL>
    ISA                 GOAL
    %STATE              DRAGDROP1-ACTION-8
  =VISUAL-STATE>
    ISA                 MODULE-STATE
    MODALITY            FREE
==>
  =GOAL>
    %STATE              DRAGDROP1-ACTION-9
  +VISUAL-LOCATION>
    ISA                 VISUAL-LOCATION
)
(P ATTEND-46
  =GOAL>
    ISA                 GOAL
    %STATE              DRAGDROP1-ACTION-9
  =VISUAL-LOCATION>
    ISA                 VISUAL-LOCATION
==>
  =GOAL>
    %STATE              DRAGDROP1-ACTION-10
  +VISUAL>
    ISA                 VISUAL-OBJECT
    SCREEN-POS          =VISUAL-LOCATION
)
(P ENCODE-47
  =GOAL>
    ISA                 GOAL
    %STATE              DRAGDROP1-ACTION-10
  =VISUAL>
    ISA                 TEXT
    VALUE               =TEXT
    VALUE               "object2"
  =VISUAL-LOCATION>
    ISA                 VISUAL-LOCATION
==>
  =GOAL>
    [TARGET]            =VISUAL-LOCATION
    %STATE              DRAGDROP1-ACTION-11
  !OUTPUT! ("Look-for-object-whose ((IS CONTENT Object2))")
)
(P POINT-TO-50
  =GOAL>
    ISA                 GOAL
    %STATE              DRAGDROP1-ACTION-11
  =MANUAL-STATE>
    ISA                 MODULE-STATE
    MODALITY            FREE
==>
  =GOAL>
    %STATE              DRAGDROP1-ACTION-12
  +MANUAL>
    ISA                 HAND-TO-MOUSE
  !OUTPUT! ("Home-to")
)
(P T51
  =GOAL>
    ISA                 GOAL
    [TARGET]            =TARGET52
    %STATE              DRAGDROP1-ACTION-12
  =MANUAL-STATE>
    ISA                 MODULE-STATE
    MODALITY            FREE
==>
  =GOAL>
    %LAST-LOCATION      =TARGET52
    %STATE              DRAGDROP1-ACTION-13
  +MANUAL>
    ISA                 MOVE-CURSOR
    LOC                 =TARGET52
  !OUTPUT! ("Point-to ~A" =TARGET52)
)
(P RELEASE-53
  =GOAL>
    ISA                 GOAL
    %STATE              DRAGDROP1-ACTION-13
  =MANUAL-STATE>
    ISA                 MODULE-STATE
    MODALITY            FREE
==>
  =GOAL>
    %STATE              DRAGDROP1-ACTION-14
  +MANUAL>
    ISA                 HAND-TO-MOUSE
  !OUTPUT! ("Home-to")
)
(P T54
  =GOAL>
    ISA                 GOAL
    %STATE              DRAGDROP1-ACTION-14
  =MANUAL-STATE>
    ISA                 MODULE-STATE
    MODALITY            FREE
==>
  =GOAL>
```

```
    %STATE              DRAGDROP1-ACTION-15
  +MANUAL>
    ISA                 RELEASE-MOUSE
)
(P RGA-56
  =GOAL>
    ISA                 GOAL
    %RETURN               =RETURN
    %STATE              DRAGDROP1-ACTION-15
==>
  =GOAL>
    %STATE              DRAGDROP1-ACTION-16
  +RETRIEVAL>
    ISA                 RECORD
    %RETURN               =RETURN
  !OUTPUT! ("RGA retrieval")
)
(P RGA-57
  =GOAL>
    ISA                 GOAL
    %STATE              DRAGDROP1-ACTION-16
  =RETRIEVAL>
    ISA                 RECORD
    %RETURN               =RETURN
    %PREVIOUS-RETURN      =PREVIOUS-RETURN
==>
  =GOAL>
    %STATE              =RETURN
    %RETURN               =PREVIOUS-RETURN
  =RETRIEVAL>
    %RETURN               #:G13373
  !OUTPUT! ("RGA")
  !EVAL! (END-TIMING 'DRAGDROP1-ACTION (PM-TIME))
)
;;;*****************************************************************
;;; Parameter settings
(SPP T43 :EFFORT 0.1) (SPP FIXED-THINK-OF-HOW-TO-DO-IT-BEST-32 :EFFORT 1.2)
;;;*****************************************************************
;;; Goal focus
(GOAL-FOCUS GOAL)
;;;And finally run:
(run-model (pathname "G2A:data; GOMS-ACTR-model.lisp ") 200)
Time  0.000: Vision found LOC63
 Time  0.000: Fixed-Think-Of-How-To-Do-It-Best-5 Selected
 FIXED-THINK-OF-HOW-TO-DO-IT-BEST-5
 IN DRAGDROP1-ACTION
 Time  1.200: Fixed-Think-Of-How-To-Do-It-Best-5 Fired
 Time  1.200: Find-7 Selected
 Time  1.250: Find-7 Fired
 Time  1.250: Module :VISION running command FIND-LOCATION
 Time  1.250: Vision found LOC63
 Time  1.250: Attend-8 Selected
 Time  1.300: Attend-8 Fired
 Time  1.300: Module :VISION running command MOVE-ATTENTION
 Time  1.385: Module :VISION running command ENCODING-COMPLETE
 Time  1.385: Vision sees TEXT62
 Time  1.385: T11 Selected
 Escape: object1 not found
 Time  1.435: T11 Fired
 Time  1.435: Find-7 Selected
 Time  1.485: Find-7 Fired
 Time  1.485: Module :VISION running command FIND-LOCATION
 Time  1.485: Vision found LOC63
 Time  1.485: Attend-8 Selected
 Time  1.535: Attend-8 Fired
 Time  1.535: Module :VISION running command MOVE-ATTENTION
 Time  1.620: Module :VISION running command ENCODING-COMPLETE
 Time  1.620: Vision sees TEXT62
 Time  1.620: T11 Selected
 Escape: object1 not found
 Time  1.670: T11 Fired
 Time  1.670: Find-7 Selected
 Time  1.720: Find-7 Fired
 Time  1.720: Module :VISION running command FIND-LOCATION
 Time  1.720: Vision found LOC66
 Time  1.720: Attend-8 Selected
 Time  1.770: Attend-8 Fired
 Time  1.770: Module :VISION running command MOVE-ATTENTION
 Time  1.855: Module :VISION running command ENCODING-COMPLETE
 Time  1.855: Vision sees TEXT61
 Time  1.855: Encode-9 Selected
 Look-for-object-whose ((IS CONTENT Object1))
 Time  1.905: Encode-9 Fired
 Time  1.905: Point-To-12 Selected
 Home-to
 Time  1.955: Point-To-12 Fired
 Time  1.955: Module :MOTOR running command HAND-TO-MOUSE
 Time  1.955: T13 Selected
 Point-to Loc66
 Time  2.005: T13 Fired
 Time  2.005: Module :MOTOR running command MOVE-CURSOR
 Time  2.205: Module :MOTOR running command PREPARATION-COMPLETE
 Time  2.255: Module :MOTOR running command INITIATION-COMPLETE
 Time  2.645: Device running command MOVE-CURSOR-ABSOLUTE
 Time  2.695: Module :MOTOR running command FINISH-MOVEMENT
 Time  2.695: Hold-Down-15 Selected
 Home-to
 Time  2.745: Hold-Down-15 Fired
 Time  2.745: Module :MOTOR running command HAND-TO-MOUSE
```

```
 Time  2.745: T16 Selected
 Time  2.845: T16 Fired
 Time  2.845: Module :MOTOR running command PRESS-MOUSE
 Time  2.845: Find-18 Selected
 Time  2.895: Find-18 Fired
 Time  2.895: Module :VISION running command FIND-LOCATION
 Time  2.895: Vision found LOC63
 Time  2.895: Attend-19 Selected
 Time  2.945: Attend-19 Fired
 Time  2.945: Module :VISION running command MOVE-ATTENTION
 Time  2.995: Module :MOTOR running command PREPARATION-COMPLETE
 Time  3.030: Module :VISION running command ENCODING-COMPLETE
 Time  3.030: Vision sees TEXT62
 Time  3.030: Encode-20 Selected
 Time  3.045: Module :MOTOR running command INITIATION-COMPLETE
 Time  3.055: Device running command OUTPUT-MOUSE-PRESS

<< Window "GOMS Window" got MOUSE PRESS at time 3055 >>
 Look-for-object-whose ((IS CONTENT Object2))
 Time  3.080: Encode-20 Fired
 Time  3.095: Module :MOTOR running command FINISH-MOVEMENT
 Time  3.095: Point-To-23 Selected
 Home-to
 Time  3.145: Point-To-23 Fired
 Time  3.145: Module :MOTOR running command HAND-TO-MOUSE
 Time  3.145: T24 Selected
 Point-to Loc63
 Time  3.195: T24 Fired
 Time  3.195: Module :MOTOR running command MOVE-CURSOR
 Time  3.395: Module :MOTOR running command PREPARATION-COMPLETE
 Time  3.445: Module :MOTOR running command INITIATION-COMPLETE
 Time  3.767: Device running command MOVE-CURSOR-ABSOLUTE
 Time  3.817: Module :MOTOR running command FINISH-MOVEMENT
 Time  3.817: Release-26 Selected
 Home-to
 Time  3.867: Release-26 Fired
 Time  3.867: Module :MOTOR running command HAND-TO-MOUSE
 Time  3.867: T27 Selected
 Time  3.917: T27 Fired
 Time  3.917: Module :MOTOR running command RELEASE-MOUSE
 Time  3.917: Rga-29 Selected
 RGA retrieval
 Time  3.967: Rga-29 Fired
 Time  3.967: Failure Retrieved
 Time  4.067: Module :MOTOR running command PREPARATION-COMPLETE
 Time  4.117: Module :MOTOR running command INITIATION-COMPLETE
 Time  4.127: Device running command OUTPUT-MOUSE-RELEASE

<< Window "GOMS Window" got MOUSE RELEASE at time 4127 >>
 Time  4.167: Module :MOTOR running command FINISH-MOVEMENT
 Time  4.167: Checking for silent events.
 Time  4.167: * Nothing to run:  No productions, no events.
4.167
4.167
[1] G2A(13):
```

# APPENDIX B: SAMPLE RUN OF G2A GENERATED MODEL USING TOISE1

```
........
drag-drop-Action-with-moveto - 416 - 181 - 224 - 25 | 343 - 186 - 224 - 25 | 0
   0.000  GOAL              SET-BUFFER-CHUNK GOAL GOAL REQUESTED NIL
   0.000  BUFFER              Buffer GOAL copied chunk GOAL to GOAL-0
   0.000  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW3-OBJECT-0 REQUESTED NIL
   0.000  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW3-OBJECT-0 to VISUAL-SW3-OBJECT-0-0
   0.000  PROCEDURAL            CONFLICT-RESOLUTION
   0.000  PROCEDURAL            PRODUCTION-SELECTED START-TASK
   0.000  PROCEDURAL            BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL            PRODUCTION-FIRED START-TASK
   0.050  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
   0.050  PROCEDURAL            CONFLICT-RESOLUTION
   0.050  PROCEDURAL            PRODUCTION-SELECTED FIND-OBJECT-C
   0.050  PROCEDURAL            BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL            QUERY-BUFFER-ACTION VISUAL
   0.100  PROCEDURAL            PRODUCTION-FIRED FIND-OBJECT-C
   0.100  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
   0.100  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
   0.100  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
   0.100  VISION            Find-location
   0.100  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW3-OBJECT-0
   0.100  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW3-OBJECT-0 to VISUAL-SW3-OBJECT-0-1
   0.100  PROCEDURAL            CONFLICT-RESOLUTION
   0.100  PROCEDURAL            PRODUCTION-SELECTED ATTEND-OBJECT-C
   0.100  PROCEDURAL            BUFFER-READ-ACTION GOAL
   0.100  PROCEDURAL            BUFFER-READ-ACTION VISUAL-LOCATION
   0.100  PROCEDURAL            QUERY-BUFFER-ACTION VISUAL
   0.150  PROCEDURAL            PRODUCTION-FIRED ATTEND-OBJECT-C
   0.150  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
   0.150  PROCEDURAL            MODULE-REQUEST VISUAL
   0.150  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
   0.150  PROCEDURAL            CLEAR-BUFFER VISUAL
   0.150  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
   0.150  VISION            Move-attention VISUAL-SW3-OBJECT-0-1 NIL
   0.150  VISION            Find-location
   0.150  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW3-OBJECT-0
   0.150  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW3-OBJECT-0 to VISUAL-SW3-OBJECT-0-2
   0.150  PROCEDURAL            CONFLICT-RESOLUTION
   0.300  VISION            Preparation-complete TRGT0
   0.300  PROCEDURAL            CONFLICT-RESOLUTION
   0.398  VISION            Complete-eye-movement TRGT0 #(174 201)
   0.398  PROCEDURAL            CONFLICT-RESOLUTION
   0.498  VISION            Preparation-complete TRGT0
   0.498  PROCEDURAL            CONFLICT-RESOLUTION
   0.572  VISION            Complete-eye-movement TRGT0 #(135 212)
   0.572  PROCEDURAL            CONFLICT-RESOLUTION
   0.602  VISION            Encoding-Complete VISUAL-SW3-OBJECT-0-1
   0.602  VISION            SET-BUFFER-CHUNK VISUAL SW3-OBJECT
   0.602  BUFFER             Buffer VISUAL copied chunk SW3-OBJECT to SW3-OBJECT-0
   0.602  PROCEDURAL            CONFLICT-RESOLUTION
   0.602  PROCEDURAL            PRODUCTION-SELECTED ENCODE-OBJECT-C
   0.602  PROCEDURAL            BUFFER-READ-ACTION GOAL
   0.602  PROCEDURAL            BUFFER-READ-ACTION VISUAL
   0.602  PROCEDURAL            BUFFER-READ-ACTION VISUAL-LOCATION
   0.652  PROCEDURAL            PRODUCTION-FIRED ENCODE-OBJECT-C
   0.652  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
   0.652  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
   0.652  PROCEDURAL            CLEAR-BUFFER VISUAL
   0.652  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
   0.652  VISION            Find-location
   0.652  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
   0.652  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-0
   0.652  PROCEDURAL            CONFLICT-RESOLUTION
   0.652  PROCEDURAL            PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-C
   0.652  PROCEDURAL            BUFFER-READ-ACTION GOAL
   0.652  PROCEDURAL            QUERY-BUFFER-ACTION MANUAL
   0.672  VISION            Preparation-complete TRGT0
   0.702  PROCEDURAL            PRODUCTION-FIRED HAND-TO-MOUSE-FOR-C
   0.702  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
   0.702  PROCEDURAL            MODULE-REQUEST MANUAL
   0.702  PROCEDURAL            CLEAR-BUFFER MANUAL
   0.702  MOTOR              HAND-TO-MOUSE
   0.702  PROCEDURAL            CONFLICT-RESOLUTION
   0.742  VISION            Complete-eye-movement TRGT0 #(135 210)
   0.742  PROCEDURAL            CONFLICT-RESOLUTION
   0.742  PROCEDURAL            PRODUCTION-SELECTED PRE-FIND-OBJECT-A
   0.742  PROCEDURAL            BUFFER-READ-ACTION GOAL
   0.742  PROCEDURAL            QUERY-BUFFER-ACTION VISUAL
   0.792  PROCEDURAL            PRODUCTION-FIRED PRE-FIND-OBJECT-A
   0.792  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
   0.792  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
   0.792  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
   0.792  VISION            Find-location
   0.792  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
   0.792  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-1
   0.792  PROCEDURAL            CONFLICT-RESOLUTION
   0.792  PROCEDURAL            PRODUCTION-SELECTED PRE-ATTEND-OBJECT-A
   0.792  PROCEDURAL            BUFFER-READ-ACTION GOAL
   0.792  PROCEDURAL            BUFFER-READ-ACTION VISUAL-LOCATION
   0.842  PROCEDURAL            PRODUCTION-FIRED PRE-ATTEND-OBJECT-A
   0.842  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
   0.842  PROCEDURAL            MODULE-REQUEST VISUAL
   0.842  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
   0.842  PROCEDURAL            CLEAR-BUFFER VISUAL
   0.842  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
   0.842  VISION            Move-attention VISUAL-SW1-OBJECT-0-1 NIL
```

```
0.842  VISION          Find-location
0.842  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
0.842  BUFFER           Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-2
0.842  PROCEDURAL          CONFLICT-RESOLUTION
0.942  VISION          Preparation-complete TRGT1
0.942  PROCEDURAL          CONFLICT-RESOLUTION
1.041  VISION          Complete-eye-movement TRGT1 #(413 197)
1.041  PROCEDURAL          CONFLICT-RESOLUTION
1.141  VISION          Preparation-complete TRGT1
1.141  PROCEDURAL          CONFLICT-RESOLUTION
1.162  VISION          Encoding-Complete VISUAL-SW1-OBJECT-0-1
1.162  VISION          SET-BUFFER-CHUNK VISUAL SW1-OBJECT
1.162  BUFFER           Buffer VISUAL copied chunk SW1-OBJECT to SW1-OBJECT-0
1.162  PROCEDURAL          CONFLICT-RESOLUTION
1.162  PROCEDURAL          PRODUCTION-SELECTED PRE-ENCODE-OBJECT-A
1.162  PROCEDURAL          BUFFER-READ-ACTION GOAL
1.162  PROCEDURAL          BUFFER-READ-ACTION VISUAL
1.162  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
1.212  PROCEDURAL          PRODUCTION-FIRED PRE-ENCODE-OBJECT-A
1.212  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
1.212  PROCEDURAL          CLEAR-BUFFER VISUAL
1.212  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
1.212  PROCEDURAL          CONFLICT-RESOLUTION
1.212  PROCEDURAL          PRODUCTION-SELECTED PRE-HAND-TO-MOUSE-FOR-A
1.212  PROCEDURAL          BUFFER-READ-ACTION GOAL
1.212  PROCEDURAL          QUERY-BUFFER-ACTION MANUAL
1.213  VISION          Complete-eye-movement TRGT1 #(413 179)
1.262  PROCEDURAL          PRODUCTION-FIRED PRE-HAND-TO-MOUSE-FOR-A
1.262  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
1.262  PROCEDURAL          MODULE-REQUEST MANUAL
1.262  PROCEDURAL          CLEAR-BUFFER MANUAL
1.262  MOTOR            HAND-TO-MOUSE
1.262  PROCEDURAL          CONFLICT-RESOLUTION
1.262  PROCEDURAL          PRODUCTION-SELECTED PRE-MOVE-MOUSE-TO-A
1.262  PROCEDURAL          BUFFER-READ-ACTION GOAL
1.262  PROCEDURAL          QUERY-BUFFER-ACTION MANUAL
1.312  PROCEDURAL          PRODUCTION-FIRED PRE-MOVE-MOUSE-TO-A
1.312  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
1.312  PROCEDURAL          MODULE-REQUEST MANUAL
1.312  PROCEDURAL          CLEAR-BUFFER MANUAL
1.312  MOTOR            MOVE-CURSOR OBJECT NIL LOC VISUAL-SW1-OBJECT-0-2
1.312  PROCEDURAL          CONFLICT-RESOLUTION
1.312  PROCEDURAL          PRODUCTION-SELECTED FIND-OBJECT-A
1.312  PROCEDURAL          BUFFER-READ-ACTION GOAL
1.312  PROCEDURAL          QUERY-BUFFER-ACTION VISUAL
1.362  PROCEDURAL          PRODUCTION-FIRED FIND-OBJECT-A
1.362  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
1.362  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
1.362  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
1.362  VISION          Find-location
1.362  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
1.362  BUFFER           Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-3
1.362  PROCEDURAL          CONFLICT-RESOLUTION
1.362  PROCEDURAL          PRODUCTION-SELECTED ATTEND-OBJECT-A
1.362  PROCEDURAL          BUFFER-READ-ACTION GOAL
1.362  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
1.362  PROCEDURAL          QUERY-BUFFER-ACTION VISUAL
1.412  PROCEDURAL          PRODUCTION-FIRED ATTEND-OBJECT-A
1.412  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
1.412  PROCEDURAL          MODULE-REQUEST VISUAL
1.412  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
1.412  PROCEDURAL          CLEAR-BUFFER VISUAL
1.412  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
1.412  VISION          Move-attention VISUAL-SW1-OBJECT-0-3 NIL
1.412  VISION          Find-location
1.412  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
1.412  BUFFER           Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-4
1.412  PROCEDURAL          CONFLICT-RESOLUTION
1.462  VISION          Preparation-complete TRGT2
1.462  PROCEDURAL          CONFLICT-RESOLUTION
1.469  VISION          Encoding-Complete VISUAL-SW1-OBJECT-0-3
1.469  VISION          SET-BUFFER-CHUNK VISUAL SW1-OBJECT
1.469  BUFFER           Buffer VISUAL copied chunk SW1-OBJECT to SW1-OBJECT-1
1.469  PROCEDURAL          CONFLICT-RESOLUTION
1.469  PROCEDURAL          PRODUCTION-SELECTED ENCODE-OBJECT-A
1.469  PROCEDURAL          BUFFER-READ-ACTION GOAL
1.469  PROCEDURAL          BUFFER-READ-ACTION VISUAL
1.469  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
1.512  MOTOR            PREPARATION-COMPLETE
1.519  PROCEDURAL          PRODUCTION-FIRED ENCODE-OBJECT-A
1.519  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
1.519  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
1.519  PROCEDURAL          CLEAR-BUFFER VISUAL
1.519  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
1.519  VISION          Find-location
1.519  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
1.519  BUFFER           Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-0
1.519  PROCEDURAL          CONFLICT-RESOLUTION
1.532  VISION          Complete-eye-movement TRGT2 #(416 181)
1.532  PROCEDURAL          CONFLICT-RESOLUTION
1.562  MOTOR            INITIATION-COMPLETE
1.562  PROCEDURAL          CONFLICT-RESOLUTION
1.662  MOTOR            MOVE-CURSOR-ABSOLUTE #(416 181)
1.662  PROCEDURAL          CONFLICT-RESOLUTION
1.712  MOTOR            FINISH-MOVEMENT
1.712  PROCEDURAL          CONFLICT-RESOLUTION
1.712  PROCEDURAL          PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-A
1.712  PROCEDURAL          BUFFER-READ-ACTION GOAL
1.712  PROCEDURAL          QUERY-BUFFER-ACTION MANUAL
```

```
1.762  PROCEDURAL        PRODUCTION-FIRED HAND-TO-MOUSE-FOR-A
1.762  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
1.762  PROCEDURAL        MODULE-REQUEST MANUAL
1.762  PROCEDURAL        CLEAR-BUFFER MANUAL
1.762  MOTOR             HAND-TO-MOUSE
1.762  PROCEDURAL        CONFLICT-RESOLUTION
1.762  PROCEDURAL        PRODUCTION-SELECTED MOVE-MOUSE-TO-A
1.762  PROCEDURAL        BUFFER-READ-ACTION GOAL
1.762  PROCEDURAL        QUERY-BUFFER-ACTION MANUAL
1.812  PROCEDURAL        PRODUCTION-FIRED MOVE-MOUSE-TO-A
1.812  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
1.812  PROCEDURAL        MODULE-REQUEST MANUAL
1.812  PROCEDURAL        CLEAR-BUFFER MANUAL
1.812  MOTOR             MOVE-CURSOR OBJECT NIL LOC VISUAL-SW1-OBJECT-0-4
1.812  PROCEDURAL        CONFLICT-RESOLUTION
1.812  PROCEDURAL        PRODUCTION-SELECTED CLICK-MOUSE-OVER-A
1.812  PROCEDURAL        BUFFER-READ-ACTION GOAL
1.812  PROCEDURAL        QUERY-BUFFER-ACTION MANUAL
1.862  PROCEDURAL        PRODUCTION-FIRED CLICK-MOUSE-OVER-A
1.862  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
1.862  PROCEDURAL        MODULE-REQUEST MANUAL
1.862  PROCEDURAL        CLEAR-BUFFER MANUAL
1.862  MOTOR             CLICK-MOUSE
1.862  PROCEDURAL        CONFLICT-RESOLUTION
1.862  PROCEDURAL        PRODUCTION-SELECTED FIND-OBJECT-B
1.862  PROCEDURAL        BUFFER-READ-ACTION GOAL
1.862  PROCEDURAL        QUERY-BUFFER-ACTION VISUAL
1.912  PROCEDURAL        PRODUCTION-FIRED FIND-OBJECT-B
1.912  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
1.912  PROCEDURAL        MODULE-REQUEST VISUAL-LOCATION
1.912  PROCEDURAL        CLEAR-BUFFER VISUAL-LOCATION
1.912  VISION            Find-location
1.912  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
1.912  BUFFER            Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-1
1.912  PROCEDURAL        CONFLICT-RESOLUTION
1.912  PROCEDURAL        PRODUCTION-SELECTED ATTEND-OBJECT-B
1.912  PROCEDURAL        BUFFER-READ-ACTION GOAL
1.912  PROCEDURAL        BUFFER-READ-ACTION VISUAL-LOCATION
1.962  PROCEDURAL        PRODUCTION-FIRED ATTEND-OBJECT-B
1.962  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
1.962  PROCEDURAL        MODULE-REQUEST VISUAL
1.962  PROCEDURAL        MODULE-REQUEST VISUAL-LOCATION
1.962  PROCEDURAL        CLEAR-BUFFER VISUAL
1.962  PROCEDURAL        CLEAR-BUFFER VISUAL-LOCATION
1.962  VISION            Move-attention VISUAL-SW2-OBJECT-0-1 NIL
1.962  VISION            Find-location
1.962  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
1.962  BUFFER            Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-2
1.962  PROCEDURAL        CONFLICT-RESOLUTION
2.012  MOTOR             PREPARATION-COMPLETE
2.012  PROCEDURAL        CONFLICT-RESOLUTION
2.062  VISION            Preparation-complete TRGT3
2.062  MOTOR             INITIATION-COMPLETE
2.062  PROCEDURAL        CONFLICT-RESOLUTION
2.072  MOTOR             OUTPUT-KEY #(28 2)
2.072  PROCEDURAL        CONFLICT-RESOLUTION
2.140  VISION            Complete-eye-movement TRGT3 #(341 178)
2.140  PROCEDURAL        CONFLICT-RESOLUTION
2.162  MOTOR             FINISH-MOVEMENT
2.162  PROCEDURAL        CONFLICT-RESOLUTION
2.214  VISION            Encoding-Complete VISUAL-SW2-OBJECT-0-1
2.214  VISION            SET-BUFFER-CHUNK VISUAL SW2-OBJECT
2.214  BUFFER            Buffer VISUAL copied chunk SW2-OBJECT to SW2-OBJECT-0
2.214  PROCEDURAL        CONFLICT-RESOLUTION
2.214  PROCEDURAL        PRODUCTION-SELECTED ENCODE-OBJECT-B
2.214  PROCEDURAL        BUFFER-READ-ACTION GOAL
2.214  PROCEDURAL        BUFFER-READ-ACTION VISUAL
2.214  PROCEDURAL        BUFFER-READ-ACTION VISUAL-LOCATION
2.240  VISION            Preparation-complete TRGT3
2.264  PROCEDURAL        PRODUCTION-FIRED ENCODE-OBJECT-B
2.264  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
2.264  PROCEDURAL        CLEAR-BUFFER VISUAL
2.264  PROCEDURAL        CLEAR-BUFFER VISUAL-LOCATION
2.264  PROCEDURAL        CONFLICT-RESOLUTION
2.264  PROCEDURAL        PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-B
2.264  PROCEDURAL        BUFFER-READ-ACTION GOAL
2.264  PROCEDURAL        QUERY-BUFFER-ACTION MANUAL
2.311  VISION            Complete-eye-movement TRGT3 #(342 188)
2.314  PROCEDURAL        PRODUCTION-FIRED HAND-TO-MOUSE-FOR-B
2.314  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
2.314  PROCEDURAL        MODULE-REQUEST MANUAL
2.314  PROCEDURAL        CLEAR-BUFFER MANUAL
2.314  MOTOR             HAND-TO-MOUSE
2.314  PROCEDURAL        CONFLICT-RESOLUTION
2.314  PROCEDURAL        PRODUCTION-SELECTED MOVE-MOUSE-TO-B
2.314  PROCEDURAL        BUFFER-READ-ACTION GOAL
2.314  PROCEDURAL        QUERY-BUFFER-ACTION MANUAL
2.364  PROCEDURAL        PRODUCTION-FIRED MOVE-MOUSE-TO-B
2.364  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
2.364  PROCEDURAL        MODULE-REQUEST MANUAL
2.364  PROCEDURAL        CLEAR-BUFFER MANUAL
2.364  MOTOR             MOVE-CURSOR OBJECT NIL LOC VISUAL-SW2-OBJECT-0-2
2.364  PROCEDURAL        CONFLICT-RESOLUTION
2.564  MOTOR             PREPARATION-COMPLETE
2.564  PROCEDURAL        CONFLICT-RESOLUTION
2.614  MOTOR             INITIATION-COMPLETE
2.614  PROCEDURAL        CONFLICT-RESOLUTION
2.714  MOTOR             MOVE-CURSOR-ABSOLUTE #(343 186)
2.714  PROCEDURAL        CONFLICT-RESOLUTION
```

```
   2.764  MOTOR            FINISH-MOVEMENT
   2.764  PROCEDURAL           CONFLICT-RESOLUTION
   2.764  PROCEDURAL           PRODUCTION-SELECTED CLICK-MOUSE-OVER-B
   2.764  PROCEDURAL           BUFFER-READ-ACTION GOAL
   2.764  PROCEDURAL           QUERY-BUFFER-ACTION MANUAL
   2.814  PROCEDURAL           PRODUCTION-FIRED CLICK-MOUSE-OVER-B
   2.814  PROCEDURAL           MODULE-REQUEST MANUAL
   2.814  PROCEDURAL           CLEAR-BUFFER GOAL
   2.814  PROCEDURAL           CLEAR-BUFFER MANUAL
   2.814  MOTOR            CLICK-MOUSE
   2.814  PROCEDURAL           CONFLICT-RESOLUTION
   2.964  MOTOR            PREPARATION-COMPLETE
   2.964  PROCEDURAL           CONFLICT-RESOLUTION
   3.014  MOTOR            INITIATION-COMPLETE
   3.014  PROCEDURAL           CONFLICT-RESOLUTION
   3.024  MOTOR            OUTPUT-KEY #(28 2)
   3.024  PROCEDURAL           CONFLICT-RESOLUTION
   3.114  MOTOR            FINISH-MOVEMENT
   3.114  PROCEDURAL           CONFLICT-RESOLUTION
   3.114  ------             Stopped because no events left to process
Execution time for drag-drop-Action-with-moveto is 3.114


drag-drop-Action-with-moveto - 259 - 49 - 150 - 16 | 135 - 210 - 237 - 267 | 0
   0.000  GOAL             SET-BUFFER-CHUNK GOAL GOAL REQUESTED NIL
   0.000  BUFFER             Buffer GOAL copied chunk GOAL to GOAL-0
   0.000  VISION           SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0 REQUESTED NIL
   0.000  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-0
   0.000  PROCEDURAL           CONFLICT-RESOLUTION
   0.000  PROCEDURAL           PRODUCTION-SELECTED START-TASK
   0.000  PROCEDURAL           BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL           PRODUCTION-FIRED START-TASK
   0.050  PROCEDURAL           MOD-BUFFER-CHUNK GOAL
   0.050  PROCEDURAL           CONFLICT-RESOLUTION
   0.050  PROCEDURAL           PRODUCTION-SELECTED FIND-OBJECT-C
   0.050  PROCEDURAL           BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL           QUERY-BUFFER-ACTION VISUAL
   0.100  PROCEDURAL           PRODUCTION-FIRED FIND-OBJECT-C
   0.100  PROCEDURAL           MOD-BUFFER-CHUNK GOAL
   0.100  PROCEDURAL           MODULE-REQUEST VISUAL-LOCATION
   0.100  PROCEDURAL           CLEAR-BUFFER VISUAL-LOCATION
   0.100  VISION           Find-location
   0.100  VISION           SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW3-OBJECT-0
   0.100  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW3-OBJECT-0 to VISUAL-SW3-OBJECT-0-0
   0.100  PROCEDURAL           CONFLICT-RESOLUTION
   0.100  PROCEDURAL           PRODUCTION-SELECTED ATTEND-OBJECT-C
   0.100  PROCEDURAL           BUFFER-READ-ACTION GOAL
   0.100  PROCEDURAL           BUFFER-READ-ACTION VISUAL-LOCATION
   0.100  PROCEDURAL           QUERY-BUFFER-ACTION VISUAL
   0.150  PROCEDURAL           PRODUCTION-FIRED ATTEND-OBJECT-C
   0.150  PROCEDURAL           MOD-BUFFER-CHUNK GOAL
   0.150  PROCEDURAL           MODULE-REQUEST VISUAL
   0.150  PROCEDURAL           MODULE-REQUEST VISUAL-LOCATION
   0.150  PROCEDURAL           CLEAR-BUFFER VISUAL
   0.150  PROCEDURAL           CLEAR-BUFFER VISUAL-LOCATION
   0.150  VISION           Move-attention VISUAL-SW3-OBJECT-0-0 NIL
   0.150  VISION           Find-location
   0.150  VISION           SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW3-OBJECT-0
   0.150  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW3-OBJECT-0 to VISUAL-SW3-OBJECT-0-1
   0.150  PROCEDURAL           CONFLICT-RESOLUTION
   0.300  VISION           Preparation-complete TRGT0
   0.300  PROCEDURAL           CONFLICT-RESOLUTION
   0.409  VISION           Complete-eye-movement TRGT0 #(210 323)
   0.409  PROCEDURAL           CONFLICT-RESOLUTION
   0.459  VISION           Preparation-complete TRGT0
   0.459  PROCEDURAL           CONFLICT-RESOLUTION
   0.535  VISION           Complete-eye-movement TRGT0 #(206 375)
   0.535  PROCEDURAL           CONFLICT-RESOLUTION
   0.583  VISION           Encoding-Complete VISUAL-SW3-OBJECT-0-0
   0.583  VISION           SET-BUFFER-CHUNK VISUAL SW3-OBJECT
   0.583  BUFFER             Buffer VISUAL copied chunk SW3-OBJECT to SW3-OBJECT-0
   0.583  PROCEDURAL           CONFLICT-RESOLUTION
   0.583  PROCEDURAL           PRODUCTION-SELECTED ENCODE-OBJECT-C
   0.583  PROCEDURAL           BUFFER-READ-ACTION GOAL
   0.583  PROCEDURAL           BUFFER-READ-ACTION VISUAL
   0.583  PROCEDURAL           BUFFER-READ-ACTION VISUAL-LOCATION
   0.633  PROCEDURAL           PRODUCTION-FIRED ENCODE-OBJECT-C
   0.633  PROCEDURAL           MOD-BUFFER-CHUNK GOAL
   0.633  PROCEDURAL           MODULE-REQUEST VISUAL-LOCATION
   0.633  PROCEDURAL           CLEAR-BUFFER VISUAL
   0.633  PROCEDURAL           CLEAR-BUFFER VISUAL-LOCATION
   0.633  VISION           Find-location
   0.633  VISION           SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
   0.633  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-1
   0.633  PROCEDURAL           CONFLICT-RESOLUTION
   0.633  PROCEDURAL           PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-C
   0.633  PROCEDURAL           BUFFER-READ-ACTION GOAL
   0.633  PROCEDURAL           QUERY-BUFFER-ACTION MANUAL
   0.635  VISION           Preparation-complete TRGT0
   0.683  PROCEDURAL           PRODUCTION-FIRED HAND-TO-MOUSE-FOR-C
   0.683  PROCEDURAL           MOD-BUFFER-CHUNK GOAL
   0.683  PROCEDURAL           MODULE-REQUEST MANUAL
   0.683  PROCEDURAL           CLEAR-BUFFER MANUAL
   0.683  MOTOR            HAND-TO-MOUSE
   0.683  PROCEDURAL           CONFLICT-RESOLUTION
   0.705  VISION           Complete-eye-movement TRGT0 #(203 378)
   0.705  PROCEDURAL           CONFLICT-RESOLUTION
   0.705  PROCEDURAL           PRODUCTION-SELECTED PRE-FIND-OBJECT-A
   0.705  PROCEDURAL           BUFFER-READ-ACTION GOAL
```

```
0.705  PROCEDURAL      QUERY-BUFFER-ACTION VISUAL
0.755  PROCEDURAL      PRODUCTION-FIRED PRE-FIND-OBJECT-A
0.755  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
0.755  PROCEDURAL      MODULE-REQUEST VISUAL-LOCATION
0.755  PROCEDURAL      CLEAR-BUFFER VISUAL-LOCATION
0.755  VISION          Find-location
0.755  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
0.755  BUFFER          Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-0
0.755  PROCEDURAL      CONFLICT-RESOLUTION
0.755  PROCEDURAL      PRODUCTION-SELECTED PRE-ATTEND-OBJECT-A
0.755  PROCEDURAL      BUFFER-READ-ACTION GOAL
0.755  PROCEDURAL      BUFFER-READ-ACTION VISUAL-LOCATION
0.805  PROCEDURAL      PRODUCTION-FIRED PRE-ATTEND-OBJECT-A
0.805  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
0.805  PROCEDURAL      MODULE-REQUEST VISUAL
0.805  PROCEDURAL      MODULE-REQUEST VISUAL-LOCATION
0.805  PROCEDURAL      CLEAR-BUFFER VISUAL
0.805  PROCEDURAL      CLEAR-BUFFER VISUAL-LOCATION
0.805  VISION          Move-attention VISUAL-SW1-OBJECT-0-0 NIL
0.805  VISION          Find-location
0.805  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
0.805  BUFFER          Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-1
0.805  PROCEDURAL      CONFLICT-RESOLUTION
0.905  VISION          Preparation-complete TRGT1
0.905  PROCEDURAL      CONFLICT-RESOLUTION
0.998  VISION          Complete-eye-movement TRGT1 #(259 163)
0.998  PROCEDURAL      CONFLICT-RESOLUTION
1.048  VISION          Preparation-complete TRGT1
1.048  PROCEDURAL      CONFLICT-RESOLUTION
1.130  VISION          Complete-eye-movement TRGT1 #(258 49)
1.130  PROCEDURAL      CONFLICT-RESOLUTION
1.179  VISION          Encoding-Complete VISUAL-SW1-OBJECT-0-0
1.179  VISION          SET-BUFFER-CHUNK VISUAL SW1-OBJECT
1.179  BUFFER          Buffer VISUAL copied chunk SW1-OBJECT to SW1-OBJECT-0
1.179  PROCEDURAL      CONFLICT-RESOLUTION
1.179  PROCEDURAL      PRODUCTION-SELECTED PRE-ENCODE-OBJECT-A
1.179  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.179  PROCEDURAL      BUFFER-READ-ACTION VISUAL
1.179  PROCEDURAL      BUFFER-READ-ACTION VISUAL-LOCATION
1.229  PROCEDURAL      PRODUCTION-FIRED PRE-ENCODE-OBJECT-A
1.229  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.229  PROCEDURAL      CLEAR-BUFFER VISUAL
1.229  PROCEDURAL      CLEAR-BUFFER VISUAL-LOCATION
1.229  PROCEDURAL      CONFLICT-RESOLUTION
1.229  PROCEDURAL      PRODUCTION-SELECTED PRE-HAND-TO-MOUSE-FOR-A
1.229  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.229  PROCEDURAL      QUERY-BUFFER-ACTION MANUAL
1.230  VISION          Preparation-complete TRGT1
1.279  PROCEDURAL      PRODUCTION-FIRED PRE-HAND-TO-MOUSE-FOR-A
1.279  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.279  PROCEDURAL      MODULE-REQUEST MANUAL
1.279  PROCEDURAL      CLEAR-BUFFER MANUAL
1.279  MOTOR           HAND-TO-MOUSE
1.279  PROCEDURAL      CONFLICT-RESOLUTION
1.279  PROCEDURAL      PRODUCTION-SELECTED PRE-MOVE-MOUSE-TO-A
1.279  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.279  PROCEDURAL      QUERY-BUFFER-ACTION MANUAL
1.300  VISION          Complete-eye-movement TRGT1 #(259 49)
1.329  PROCEDURAL      PRODUCTION-FIRED PRE-MOVE-MOUSE-TO-A
1.329  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.329  PROCEDURAL      MODULE-REQUEST MANUAL
1.329  PROCEDURAL      CLEAR-BUFFER MANUAL
1.329  MOTOR           MOVE-CURSOR OBJECT NIL LOC VISUAL-SW1-OBJECT-0-1
1.329  PROCEDURAL      CONFLICT-RESOLUTION
1.329  PROCEDURAL      PRODUCTION-SELECTED FIND-OBJECT-A
1.329  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.329  PROCEDURAL      QUERY-BUFFER-ACTION VISUAL
1.379  PROCEDURAL      PRODUCTION-FIRED FIND-OBJECT-A
1.379  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.379  PROCEDURAL      MODULE-REQUEST VISUAL-LOCATION
1.379  PROCEDURAL      CLEAR-BUFFER VISUAL-LOCATION
1.379  VISION          Find-location
1.379  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
1.379  BUFFER          Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-2
1.379  PROCEDURAL      CONFLICT-RESOLUTION
1.379  PROCEDURAL      PRODUCTION-SELECTED ATTEND-OBJECT-A
1.379  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.379  PROCEDURAL      BUFFER-READ-ACTION VISUAL-LOCATION
1.379  PROCEDURAL      QUERY-BUFFER-ACTION VISUAL
1.429  PROCEDURAL      PRODUCTION-FIRED ATTEND-OBJECT-A
1.429  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.429  PROCEDURAL      MODULE-REQUEST VISUAL
1.429  PROCEDURAL      MODULE-REQUEST VISUAL-LOCATION
1.429  PROCEDURAL      CLEAR-BUFFER VISUAL
1.429  PROCEDURAL      CLEAR-BUFFER VISUAL-LOCATION
1.429  VISION          Move-attention VISUAL-SW1-OBJECT-0-2 NIL
1.429  VISION          Find-location
1.429  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
1.429  BUFFER          Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-3
1.429  VISION          Preparation-complete TRGT2
1.429  PROCEDURAL      CONFLICT-RESOLUTION
1.475  VISION          Encoding-Complete VISUAL-SW1-OBJECT-0-2
1.475  VISION          SET-BUFFER-CHUNK VISUAL SW1-OBJECT
1.475  BUFFER          Buffer VISUAL copied chunk SW1-OBJECT to SW1-OBJECT-1
1.475  PROCEDURAL      CONFLICT-RESOLUTION
1.475  PROCEDURAL      PRODUCTION-SELECTED ENCODE-OBJECT-A
1.475  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.475  PROCEDURAL      BUFFER-READ-ACTION VISUAL
1.475  PROCEDURAL      BUFFER-READ-ACTION VISUAL-LOCATION
```

```
1.499  VISION       Complete-eye-movement TRGT2 #(259 49)
1.525  PROCEDURAL       PRODUCTION-FIRED ENCODE-OBJECT-A
1.525  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
1.525  PROCEDURAL       MODULE-REQUEST VISUAL-LOCATION
1.525  PROCEDURAL       CLEAR-BUFFER VISUAL
1.525  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
1.525  VISION       Find-location
1.525  VISION       SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
1.525  BUFFER        Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-2
1.525  PROCEDURAL       CONFLICT-RESOLUTION
1.529  MOTOR        PREPARATION-COMPLETE
1.529  PROCEDURAL       CONFLICT-RESOLUTION
1.579  MOTOR        INITIATION-COMPLETE
1.579  PROCEDURAL       CONFLICT-RESOLUTION
2.015  MOTOR        MOVE-CURSOR-ABSOLUTE #(259 49)
2.015  PROCEDURAL       CONFLICT-RESOLUTION
2.065  MOTOR        FINISH-MOVEMENT
2.065  PROCEDURAL       CONFLICT-RESOLUTION
2.065  PROCEDURAL       PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-A
2.065  PROCEDURAL       BUFFER-READ-ACTION GOAL
2.065  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
2.115  PROCEDURAL       PRODUCTION-FIRED HAND-TO-MOUSE-FOR-A
2.115  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
2.115  PROCEDURAL       MODULE-REQUEST MANUAL
2.115  PROCEDURAL       CLEAR-BUFFER MANUAL
2.115  MOTOR        HAND-TO-MOUSE
2.115  PROCEDURAL       CONFLICT-RESOLUTION
2.115  PROCEDURAL       PRODUCTION-SELECTED MOVE-MOUSE-TO-A
2.115  PROCEDURAL       BUFFER-READ-ACTION GOAL
2.115  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
2.165  PROCEDURAL       PRODUCTION-FIRED MOVE-MOUSE-TO-A
2.165  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
2.165  PROCEDURAL       MODULE-REQUEST MANUAL
2.165  PROCEDURAL       CLEAR-BUFFER MANUAL
2.165  MOTOR        MOVE-CURSOR OBJECT NIL LOC VISUAL-SW1-OBJECT-0-3
2.165  PROCEDURAL       CONFLICT-RESOLUTION
2.165  PROCEDURAL       PRODUCTION-SELECTED CLICK-MOUSE-OVER-A
2.165  PROCEDURAL       BUFFER-READ-ACTION GOAL
2.165  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
2.215  PROCEDURAL       PRODUCTION-FIRED CLICK-MOUSE-OVER-A
2.215  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
2.215  PROCEDURAL       MODULE-REQUEST MANUAL
2.215  PROCEDURAL       CLEAR-BUFFER MANUAL
2.215  MOTOR        CLICK-MOUSE
2.215  PROCEDURAL       CONFLICT-RESOLUTION
2.215  PROCEDURAL       PRODUCTION-SELECTED FIND-OBJECT-B
2.215  PROCEDURAL       BUFFER-READ-ACTION GOAL
2.215  PROCEDURAL       QUERY-BUFFER-ACTION VISUAL
2.265  PROCEDURAL       PRODUCTION-FIRED FIND-OBJECT-B
2.265  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
2.265  PROCEDURAL       MODULE-REQUEST VISUAL-LOCATION
2.265  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
2.265  VISION       Find-location
2.265  VISION       SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
2.265  BUFFER        Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-3
2.265  PROCEDURAL       CONFLICT-RESOLUTION
2.265  PROCEDURAL       PRODUCTION-SELECTED ATTEND-OBJECT-B
2.265  PROCEDURAL       BUFFER-READ-ACTION GOAL
2.265  PROCEDURAL       BUFFER-READ-ACTION VISUAL-LOCATION
2.315  PROCEDURAL       PRODUCTION-FIRED ATTEND-OBJECT-B
2.315  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
2.315  PROCEDURAL       MODULE-REQUEST VISUAL
2.315  PROCEDURAL       MODULE-REQUEST VISUAL-LOCATION
2.315  PROCEDURAL       CLEAR-BUFFER VISUAL
2.315  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
2.315  VISION       Move-attention VISUAL-SW2-OBJECT-0-3 NIL
2.315  VISION       Find-location
2.315  VISION       SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
2.315  BUFFER        Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-4
2.315  PROCEDURAL       CONFLICT-RESOLUTION
2.365  MOTOR        PREPARATION-COMPLETE
2.365  PROCEDURAL       CONFLICT-RESOLUTION
2.415  VISION       Preparation-complete TRGT3
2.415  MOTOR        INITIATION-COMPLETE
2.415  PROCEDURAL       CONFLICT-RESOLUTION
2.425  MOTOR        OUTPUT-KEY #(28 2)
2.425  PROCEDURAL       CONFLICT-RESOLUTION
2.506  VISION       Complete-eye-movement TRGT3 #(126 204)
2.506  PROCEDURAL       CONFLICT-RESOLUTION
2.515  MOTOR        FINISH-MOVEMENT
2.515  PROCEDURAL       CONFLICT-RESOLUTION
2.583  VISION       Encoding-Complete VISUAL-SW2-OBJECT-0-3
2.583  VISION       SET-BUFFER-CHUNK VISUAL SW2-OBJECT
2.583  BUFFER        Buffer VISUAL copied chunk SW2-OBJECT to SW2-OBJECT-0
2.583  PROCEDURAL       CONFLICT-RESOLUTION
2.583  PROCEDURAL       PRODUCTION-SELECTED ENCODE-OBJECT-B
2.583  PROCEDURAL       BUFFER-READ-ACTION GOAL
2.583  PROCEDURAL       BUFFER-READ-ACTION VISUAL
2.583  PROCEDURAL       BUFFER-READ-ACTION VISUAL-LOCATION
2.606  VISION       Preparation-complete TRGT3
2.633  PROCEDURAL       PRODUCTION-FIRED ENCODE-OBJECT-B
2.633  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
2.633  PROCEDURAL       CLEAR-BUFFER VISUAL
2.633  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
2.633  PROCEDURAL       CONFLICT-RESOLUTION
2.633  PROCEDURAL       PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-B
2.633  PROCEDURAL       BUFFER-READ-ACTION GOAL
2.633  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
2.677  VISION       Complete-eye-movement TRGT3 #(134 210)
```

```
   2.683  PROCEDURAL          PRODUCTION-FIRED HAND-TO-MOUSE-FOR-B
   2.683  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   2.683  PROCEDURAL          MODULE-REQUEST MANUAL
   2.683  PROCEDURAL          CLEAR-BUFFER MANUAL
   2.683  MOTOR               HAND-TO-MOUSE
   2.683  PROCEDURAL          CONFLICT-RESOLUTION
   2.683  PROCEDURAL          PRODUCTION-SELECTED MOVE-MOUSE-TO-B
   2.683  PROCEDURAL          BUFFER-READ-ACTION GOAL
   2.683  PROCEDURAL          QUERY-BUFFER-ACTION MANUAL
   2.733  PROCEDURAL          PRODUCTION-FIRED MOVE-MOUSE-TO-B
   2.733  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   2.733  PROCEDURAL          MODULE-REQUEST MANUAL
   2.733  PROCEDURAL          CLEAR-BUFFER MANUAL
   2.733  MOTOR               MOVE-CURSOR OBJECT NIL LOC VISUAL-SW2-OBJECT-0-4
   2.733  PROCEDURAL          CONFLICT-RESOLUTION
   2.933  MOTOR               PREPARATION-COMPLETE
   2.933  PROCEDURAL          CONFLICT-RESOLUTION
   2.983  MOTOR               INITIATION-COMPLETE
   2.983  PROCEDURAL          CONFLICT-RESOLUTION
   3.083  MOTOR               MOVE-CURSOR-ABSOLUTE #(135 210)
   3.083  PROCEDURAL          CONFLICT-RESOLUTION
   3.133  MOTOR               FINISH-MOVEMENT
   3.133  PROCEDURAL          CONFLICT-RESOLUTION
   3.133  PROCEDURAL          PRODUCTION-SELECTED CLICK-MOUSE-OVER-B
   3.133  PROCEDURAL          BUFFER-READ-ACTION GOAL
   3.133  PROCEDURAL          QUERY-BUFFER-ACTION MANUAL
   3.183  PROCEDURAL          PRODUCTION-FIRED CLICK-MOUSE-OVER-B
   3.183  PROCEDURAL          MODULE-REQUEST MANUAL
   3.183  PROCEDURAL          CLEAR-BUFFER GOAL
   3.183  PROCEDURAL          CLEAR-BUFFER MANUAL
   3.183  MOTOR               CLICK-MOUSE
   3.183  PROCEDURAL          CONFLICT-RESOLUTION
   3.333  MOTOR               PREPARATION-COMPLETE
   3.333  PROCEDURAL          CONFLICT-RESOLUTION
   3.383  MOTOR               INITIATION-COMPLETE
   3.383  PROCEDURAL          CONFLICT-RESOLUTION
   3.393  MOTOR               OUTPUT-KEY #(28 2)
   3.393  PROCEDURAL          CONFLICT-RESOLUTION
   3.483  MOTOR               FINISH-MOVEMENT
   3.483  PROCEDURAL          CONFLICT-RESOLUTION
   3.483  ------              Stopped because no events left to process
Execution time for drag-drop-Action-with-moveto is 3.483


click-action-with-moveto - 343 - 186 - 224 - 25 - 202 - 379 - 131 - 23 | 0
   0.000  GOAL                SET-BUFFER-CHUNK GOAL GOAL REQUESTED NIL
   0.000  BUFFER              Buffer GOAL copied chunk GOAL to GOAL-0
   0.000  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0 REQUESTED NIL
   0.000  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-0
   0.000  PROCEDURAL          CONFLICT-RESOLUTION
   0.000  PROCEDURAL          PRODUCTION-SELECTED START-TASK
   0.000  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL          PRODUCTION-FIRED START-TASK
   0.050  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.050  PROCEDURAL          CONFLICT-RESOLUTION
   0.050  PROCEDURAL          PRODUCTION-SELECTED FIND-OBJECT-A
   0.050  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL          QUERY-BUFFER-ACTION VISUAL
   0.100  PROCEDURAL          PRODUCTION-FIRED FIND-OBJECT-A
   0.100  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.100  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
   0.100  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
   0.100  VISION              Find-location
   0.100  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
   0.100  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-0
   0.100  PROCEDURAL          CONFLICT-RESOLUTION
   0.100  PROCEDURAL          PRODUCTION-SELECTED ATTEND-OBJECT-A
   0.100  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.100  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
   0.100  PROCEDURAL          QUERY-BUFFER-ACTION VISUAL
   0.150  PROCEDURAL          PRODUCTION-FIRED ATTEND-OBJECT-A
   0.150  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.150  PROCEDURAL          MODULE-REQUEST VISUAL
   0.150  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
   0.150  PROCEDURAL          CLEAR-BUFFER VISUAL
   0.150  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
   0.150  VISION              Move-attention VISUAL-SW1-OBJECT-0-0 NIL
   0.150  VISION              Find-location
   0.150  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
   0.150  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-1
   0.150  PROCEDURAL          CONFLICT-RESOLUTION
   0.300  VISION              Preparation-complete TRGT0
   0.300  PROCEDURAL          CONFLICT-RESOLUTION
   0.409  VISION              Complete-eye-movement TRGT0 #(335 178)
   0.409  PROCEDURAL          CONFLICT-RESOLUTION
   0.459  VISION              Preparation-complete TRGT0
   0.459  PROCEDURAL          CONFLICT-RESOLUTION
   0.498  VISION              Encoding-Complete VISUAL-SW1-OBJECT-0-0
   0.498  VISION              SET-BUFFER-CHUNK VISUAL SW1-OBJECT
   0.498  BUFFER              Buffer VISUAL copied chunk SW1-OBJECT to SW1-OBJECT-0
   0.498  PROCEDURAL          CONFLICT-RESOLUTION
   0.498  PROCEDURAL          PRODUCTION-SELECTED ENCODE-OBJECT-A
   0.498  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.498  PROCEDURAL          BUFFER-READ-ACTION VISUAL
   0.498  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
   0.530  VISION              Complete-eye-movement TRGT0 #(344 185)
   0.548  PROCEDURAL          PRODUCTION-FIRED ENCODE-OBJECT-A
   0.548  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.548  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
```

```
0.548  PROCEDURAL       CLEAR-BUFFER VISUAL
0.548  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
0.548  VISION           Find-location
0.548  VISION           SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
0.548  BUFFER           Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-1
0.548  PROCEDURAL       CONFLICT-RESOLUTION
0.548  PROCEDURAL       PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-A
0.548  PROCEDURAL       BUFFER-READ-ACTION GOAL
0.548  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
0.598  PROCEDURAL       PRODUCTION-FIRED HAND-TO-MOUSE-FOR-A
0.598  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
0.598  PROCEDURAL       MODULE-REQUEST MANUAL
0.598  PROCEDURAL       CLEAR-BUFFER MANUAL
0.598  MOTOR            HAND-TO-MOUSE
0.598  PROCEDURAL       CONFLICT-RESOLUTION
0.598  PROCEDURAL       PRODUCTION-SELECTED FIND-OBJECT-B
0.598  PROCEDURAL       BUFFER-READ-ACTION GOAL
0.598  PROCEDURAL       QUERY-BUFFER-ACTION VISUAL
0.648  PROCEDURAL       PRODUCTION-FIRED FIND-OBJECT-B
0.648  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
0.648  PROCEDURAL       MODULE-REQUEST VISUAL-LOCATION
0.648  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
0.648  VISION           Find-location
0.648  VISION           SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
0.648  BUFFER           Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-2
0.648  PROCEDURAL       CONFLICT-RESOLUTION
0.648  PROCEDURAL       PRODUCTION-SELECTED ATTEND-OBJECT-B
0.648  PROCEDURAL       BUFFER-READ-ACTION GOAL
0.648  PROCEDURAL       BUFFER-READ-ACTION VISUAL-LOCATION
0.698  PROCEDURAL       PRODUCTION-FIRED ATTEND-OBJECT-B
0.698  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
0.698  PROCEDURAL       MODULE-REQUEST VISUAL
0.698  PROCEDURAL       MODULE-REQUEST VISUAL-LOCATION
0.698  PROCEDURAL       CLEAR-BUFFER VISUAL
0.698  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
0.698  VISION           Move-attention VISUAL-SW2-OBJECT-0-2 NIL
0.698  VISION           Find-location
0.698  VISION           SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
0.698  BUFFER           Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-3
0.698  PROCEDURAL       CONFLICT-RESOLUTION
0.798  VISION           Preparation-complete TRGT1
0.798  PROCEDURAL       CONFLICT-RESOLUTION
0.892  VISION           Complete-eye-movement TRGT1 #(190 352)
0.892  PROCEDURAL       CONFLICT-RESOLUTION
0.992  VISION           Preparation-complete TRGT1
0.992  PROCEDURAL       CONFLICT-RESOLUTION
1.065  VISION           Complete-eye-movement TRGT1 #(202 375)
1.065  VISION           Preparation-complete TRGT1
1.065  PROCEDURAL       CONFLICT-RESOLUTION
1.068  VISION           Encoding-Complete VISUAL-SW2-OBJECT-0-2
1.068  VISION           SET-BUFFER-CHUNK VISUAL SW2-OBJECT
1.068  BUFFER           Buffer VISUAL copied chunk SW2-OBJECT to SW2-OBJECT-0
1.068  PROCEDURAL       CONFLICT-RESOLUTION
1.068  PROCEDURAL       PRODUCTION-SELECTED ENCODE-OBJECT-B
1.068  PROCEDURAL       BUFFER-READ-ACTION GOAL
1.068  PROCEDURAL       BUFFER-READ-ACTION VISUAL
1.068  PROCEDURAL       BUFFER-READ-ACTION VISUAL-LOCATION
1.118  PROCEDURAL       PRODUCTION-FIRED ENCODE-OBJECT-B
1.118  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
1.118  PROCEDURAL       CLEAR-BUFFER VISUAL
1.118  PROCEDURAL       CLEAR-BUFFER VISUAL-LOCATION
1.118  PROCEDURAL       CONFLICT-RESOLUTION
1.118  PROCEDURAL       PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-B
1.118  PROCEDURAL       BUFFER-READ-ACTION GOAL
1.118  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
1.135  VISION           Complete-eye-movement TRGT1 #(202 379)
1.168  PROCEDURAL       PRODUCTION-FIRED HAND-TO-MOUSE-FOR-B
1.168  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
1.168  PROCEDURAL       MODULE-REQUEST MANUAL
1.168  PROCEDURAL       CLEAR-BUFFER MANUAL
1.168  MOTOR            HAND-TO-MOUSE
1.168  PROCEDURAL       CONFLICT-RESOLUTION
1.168  PROCEDURAL       PRODUCTION-SELECTED MOVE-MOUSE-TO-B
1.168  PROCEDURAL       BUFFER-READ-ACTION GOAL
1.168  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
1.218  PROCEDURAL       PRODUCTION-FIRED MOVE-MOUSE-TO-B
1.218  PROCEDURAL       MOD-BUFFER-CHUNK GOAL
1.218  PROCEDURAL       MODULE-REQUEST MANUAL
1.218  PROCEDURAL       CLEAR-BUFFER MANUAL
1.218  MOTOR            MOVE-CURSOR OBJECT NIL LOC VISUAL-SW2-OBJECT-0-3
1.218  PROCEDURAL       CONFLICT-RESOLUTION
1.418  MOTOR            PREPARATION-COMPLETE
1.418  PROCEDURAL       CONFLICT-RESOLUTION
1.468  MOTOR            INITIATION-COMPLETE
1.468  PROCEDURAL       CONFLICT-RESOLUTION
1.781  MOTOR            MOVE-CURSOR-ABSOLUTE #(202 379)
1.781  PROCEDURAL       CONFLICT-RESOLUTION
1.831  MOTOR            FINISH-MOVEMENT
1.831  PROCEDURAL       CONFLICT-RESOLUTION
1.831  PROCEDURAL       PRODUCTION-SELECTED CLICK-MOUSE-OVER-B
1.831  PROCEDURAL       BUFFER-READ-ACTION GOAL
1.831  PROCEDURAL       QUERY-BUFFER-ACTION MANUAL
1.881  PROCEDURAL       PRODUCTION-FIRED CLICK-MOUSE-OVER-B
1.881  PROCEDURAL       MODULE-REQUEST MANUAL
1.881  PROCEDURAL       CLEAR-BUFFER GOAL
1.881  PROCEDURAL       CLEAR-BUFFER MANUAL
1.881  MOTOR            CLICK-MOUSE
1.881  PROCEDURAL       CONFLICT-RESOLUTION
2.031  MOTOR            PREPARATION-COMPLETE
```

```
   2.031  PROCEDURAL          CONFLICT-RESOLUTION
   2.081  MOTOR              INITIATION-COMPLETE
   2.081  PROCEDURAL          CONFLICT-RESOLUTION
   2.091  MOTOR              OUTPUT-KEY #(28 2)
   2.091  PROCEDURAL          CONFLICT-RESOLUTION
   2.181  MOTOR              FINISH-MOVEMENT
   2.181  PROCEDURAL          CONFLICT-RESOLUTION
   2.181  ------              Stopped because no events left to process
Execution time for click-action-with-moveto is 2.181


click-action-with-moveto - 202 - 379 - 131 - 23 - 571 - 293 - 51 - 23 | 0
   0.000  GOAL               SET-BUFFER-CHUNK GOAL GOAL REQUESTED NIL
   0.000  BUFFER             Buffer GOAL copied chunk GOAL to GOAL-0
   0.000  VISION             SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0 REQUESTED NIL
   0.000  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-0
   0.000  PROCEDURAL          CONFLICT-RESOLUTION
   0.000  PROCEDURAL          PRODUCTION-SELECTED START-TASK
   0.000  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL          PRODUCTION-FIRED START-TASK
   0.050  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.050  PROCEDURAL          CONFLICT-RESOLUTION
   0.050  PROCEDURAL          PRODUCTION-SELECTED FIND-OBJECT-A
   0.050  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.050  PROCEDURAL          QUERY-BUFFER-ACTION VISUAL
   0.100  PROCEDURAL          PRODUCTION-FIRED FIND-OBJECT-A
   0.100  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.100  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
   0.100  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
   0.100  VISION              Find-location
   0.100  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
   0.100  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-1
   0.100  PROCEDURAL          CONFLICT-RESOLUTION
   0.100  PROCEDURAL          PRODUCTION-SELECTED ATTEND-OBJECT-A
   0.100  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.100  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
   0.100  PROCEDURAL          QUERY-BUFFER-ACTION VISUAL
   0.150  PROCEDURAL          PRODUCTION-FIRED ATTEND-OBJECT-A
   0.150  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.150  PROCEDURAL          MODULE-REQUEST VISUAL
   0.150  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
   0.150  PROCEDURAL          CLEAR-BUFFER VISUAL
   0.150  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
   0.150  VISION              Move-attention VISUAL-SW1-OBJECT-0-1 NIL
   0.150  VISION              Find-location
   0.150  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
   0.150  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-2
   0.150  PROCEDURAL          CONFLICT-RESOLUTION
   0.300  VISION              Preparation-complete TRGT0
   0.300  PROCEDURAL          CONFLICT-RESOLUTION
   0.406  VISION              Complete-eye-movement TRGT0 #(130 323)
   0.406  PROCEDURAL          CONFLICT-RESOLUTION
   0.456  VISION              Preparation-complete TRGT0
   0.456  PROCEDURAL          CONFLICT-RESOLUTION
   0.536  VISION              Complete-eye-movement TRGT0 #(207 382)
   0.536  PROCEDURAL          CONFLICT-RESOLUTION
   0.599  VISION              Encoding-Complete VISUAL-SW1-OBJECT-0-1
   0.599  VISION              SET-BUFFER-CHUNK VISUAL SW1-OBJECT
   0.599  BUFFER              Buffer VISUAL copied chunk SW1-OBJECT to SW1-OBJECT-0
   0.599  PROCEDURAL          CONFLICT-RESOLUTION
   0.599  PROCEDURAL          PRODUCTION-SELECTED ENCODE-OBJECT-A
   0.599  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.599  PROCEDURAL          BUFFER-READ-ACTION VISUAL
   0.599  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
   0.636  VISION              Preparation-complete TRGT0
   0.649  PROCEDURAL          PRODUCTION-FIRED ENCODE-OBJECT-A
   0.649  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.649  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
   0.649  PROCEDURAL          CLEAR-BUFFER VISUAL
   0.649  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
   0.649  VISION              Find-location
   0.649  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
   0.649  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-0
   0.649  PROCEDURAL          CONFLICT-RESOLUTION
   0.649  PROCEDURAL          PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-A
   0.649  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.649  PROCEDURAL          QUERY-BUFFER-ACTION MANUAL
   0.699  PROCEDURAL          PRODUCTION-FIRED HAND-TO-MOUSE-FOR-A
   0.699  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.699  PROCEDURAL          MODULE-REQUEST MANUAL
   0.699  PROCEDURAL          CLEAR-BUFFER MANUAL
   0.699  MOTOR              HAND-TO-MOUSE
   0.699  PROCEDURAL          CONFLICT-RESOLUTION
   0.707  VISION              Complete-eye-movement TRGT0 #(202 379)
   0.707  PROCEDURAL          CONFLICT-RESOLUTION
   0.707  PROCEDURAL          PRODUCTION-SELECTED FIND-OBJECT-B
   0.707  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.707  PROCEDURAL          QUERY-BUFFER-ACTION VISUAL
   0.757  PROCEDURAL          PRODUCTION-FIRED FIND-OBJECT-B
   0.757  PROCEDURAL          MOD-BUFFER-CHUNK GOAL
   0.757  PROCEDURAL          MODULE-REQUEST VISUAL-LOCATION
   0.757  PROCEDURAL          CLEAR-BUFFER VISUAL-LOCATION
   0.757  VISION              Find-location
   0.757  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
   0.757  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-1
   0.757  PROCEDURAL          CONFLICT-RESOLUTION
   0.757  PROCEDURAL          PRODUCTION-SELECTED ATTEND-OBJECT-B
   0.757  PROCEDURAL          BUFFER-READ-ACTION GOAL
   0.757  PROCEDURAL          BUFFER-READ-ACTION VISUAL-LOCATION
```

```
0.807  PROCEDURAL      PRODUCTION-FIRED ATTEND-OBJECT-B
0.807  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
0.807  PROCEDURAL      MODULE-REQUEST VISUAL
0.807  PROCEDURAL      MODULE-REQUEST VISUAL-LOCATION
0.807  PROCEDURAL      CLEAR-BUFFER VISUAL
0.807  PROCEDURAL      CLEAR-BUFFER VISUAL-LOCATION
0.807  VISION          Move-attention VISUAL-SW2-OBJECT-0-1 NIL
0.807  VISION          Find-location
0.807  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
0.807  BUFFER          Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-2
0.807  PROCEDURAL      CONFLICT-RESOLUTION
0.907  VISION          Preparation-complete TRGT1
0.907  PROCEDURAL      CONFLICT-RESOLUTION
1.021  VISION          Complete-eye-movement TRGT1 #(628 285)
1.021  PROCEDURAL      CONFLICT-RESOLUTION
1.121  VISION          Preparation-complete TRGT1
1.121  PROCEDURAL      CONFLICT-RESOLUTION
1.197  VISION          Complete-eye-movement TRGT1 #(574 293)
1.197  PROCEDURAL      CONFLICT-RESOLUTION
1.241  VISION          Encoding-Complete VISUAL-SW2-OBJECT-0-1
1.241  VISION          SET-BUFFER-CHUNK VISUAL SW2-OBJECT
1.241  BUFFER          Buffer VISUAL copied chunk SW2-OBJECT to SW2-OBJECT-0
1.241  PROCEDURAL      CONFLICT-RESOLUTION
1.241  PROCEDURAL      PRODUCTION-SELECTED ENCODE-OBJECT-B
1.241  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.241  PROCEDURAL      BUFFER-READ-ACTION VISUAL
1.241  PROCEDURAL      BUFFER-READ-ACTION VISUAL-LOCATION
1.247  VISION          Preparation-complete TRGT1
1.291  PROCEDURAL      PRODUCTION-FIRED ENCODE-OBJECT-B
1.291  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.291  PROCEDURAL      CLEAR-BUFFER VISUAL
1.291  PROCEDURAL      CLEAR-BUFFER VISUAL-LOCATION
1.291  PROCEDURAL      CONFLICT-RESOLUTION
1.291  PROCEDURAL      PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-B
1.291  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.291  PROCEDURAL      QUERY-BUFFER-ACTION MANUAL
1.317  VISION          Complete-eye-movement TRGT1 #(571 293)
1.341  PROCEDURAL      PRODUCTION-FIRED HAND-TO-MOUSE-FOR-B
1.341  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.341  PROCEDURAL      MODULE-REQUEST MANUAL
1.341  PROCEDURAL      CLEAR-BUFFER MANUAL
1.341  MOTOR           HAND-TO-MOUSE
1.341  PROCEDURAL      CONFLICT-RESOLUTION
1.341  PROCEDURAL      PRODUCTION-SELECTED MOVE-MOUSE-TO-B
1.341  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.341  PROCEDURAL      QUERY-BUFFER-ACTION MANUAL
1.391  PROCEDURAL      PRODUCTION-FIRED MOVE-MOUSE-TO-B
1.391  PROCEDURAL      MOD-BUFFER-CHUNK GOAL
1.391  PROCEDURAL      MODULE-REQUEST MANUAL
1.391  PROCEDURAL      CLEAR-BUFFER MANUAL
1.391  MOTOR           MOVE-CURSOR OBJECT NIL LOC VISUAL-SW2-OBJECT-0-2
1.391  PROCEDURAL      CONFLICT-RESOLUTION
1.591  MOTOR           PREPARATION-COMPLETE
1.591  PROCEDURAL      CONFLICT-RESOLUTION
1.641  MOTOR           INITIATION-COMPLETE
1.641  PROCEDURAL      CONFLICT-RESOLUTION
1.931  MOTOR           MOVE-CURSOR-ABSOLUTE #(571 293)
1.931  PROCEDURAL      CONFLICT-RESOLUTION
1.981  MOTOR           FINISH-MOVEMENT
1.981  PROCEDURAL      CONFLICT-RESOLUTION
1.981  PROCEDURAL      PRODUCTION-SELECTED CLICK-MOUSE-OVER-B
1.981  PROCEDURAL      BUFFER-READ-ACTION GOAL
1.981  PROCEDURAL      QUERY-BUFFER-ACTION MANUAL
2.031  PROCEDURAL      PRODUCTION-FIRED CLICK-MOUSE-OVER-B
2.031  PROCEDURAL      MODULE-REQUEST MANUAL
2.031  PROCEDURAL      CLEAR-BUFFER GOAL
2.031  PROCEDURAL      CLEAR-BUFFER MANUAL
2.031  MOTOR           CLICK-MOUSE
2.031  PROCEDURAL      CONFLICT-RESOLUTION
2.181  MOTOR           PREPARATION-COMPLETE
2.181  PROCEDURAL      CONFLICT-RESOLUTION
2.231  MOTOR           INITIATION-COMPLETE
2.231  PROCEDURAL      CONFLICT-RESOLUTION
2.241  MOTOR           OUTPUT-KEY #(28 2)
2.241  PROCEDURAL      CONFLICT-RESOLUTION
2.331  MOTOR           FINISH-MOVEMENT
2.331  PROCEDURAL      CONFLICT-RESOLUTION
2.331  ------          Stopped because no events left to process
Execution time for click-action-with-moveto is 2.331
```

# APPENDIX C: AN EXAMPLE OF MACROS TO RUN ACT-R INTERACTION PRIMITIVE DRAG&DRAP

```
(defmacro drag-drop-Action-macro(obj1X obj1Y obj1W obj1H obj2X obj2Y obj2W obj2H thinking)
  `(progn
        (reset)
        (define-model TESTMODEL
        (set-cursor-position-fct *last-cursor-position*)
        (start-hand-at-mouse)
        (set-cursor-position-fct *last-cursor-position*)
        (chunk-type (VISUAL-ITEM  (:include visual-object) )CONTENT)
        (chunk-type IDTR-ACTION %ID)
        (chunk-type (a-object-location (:include visual-location)))
        (chunk-type (b-object-location (:include visual-location)))
        (chunk-type (a-object (:include visual-object)) CONTENT)
        (chunk-type (b-object (:include visual-object)) CONTENT)
        (chunk-type GOAL STATE %LAST-LOCATION [SRC-TARGET] [DEST-TARGET])
        (define-chunks (a-object isa chunk))
        (define-chunks (b-object isa chunk))
        (add-dm

       (FIND-OBJECT-A-ACTION  ISA IDTR-ACTION )  (ATTEND-OBJECT-A-ACTION  ISA IDTR-ACTION )
       (ENCODE-OBJECT-A-ACTION  ISA IDTR-ACTION )   (HAND-TO-MOUSE-FOR-A-ACTION  ISA IDTR-ACTION )
       (MOUSE-MOVES-A-ACTIONS  ISA IDTR-ACTION ) (ATTEND-OBJECT-A-ACTION4 ISA IDTR-ACTION )
       (CLICK-MOUSE-OVER-A-ACTION  ISA IDTR-ACTION )   (FIND-OBJECT-B-ACTION  ISA IDTR-ACTION )
       (ATTEND-OBJECT-B-ACTION  ISA IDTR-ACTION )   (ENCODE-OBJECT-B-ACTION  ISA IDTR-ACTION )
       (HAND-TO-MOUSE-FOR-B-ACTION ISA IDTR-ACTION )   (MOVE-MOUSE-TO-B-ACTION ISA IDTR-ACTION )
       (ATTEND-OBJECT-A-ACTION2 ISA IDTR-ACTION )   (MOUSE-MOVES-B-ACTIONS ISA IDTR-ACTION )
       (START-ACTION ISA IDTR-ACTION %ID START-ACTION)
       (THINKING ISA IDTR-ACTION %ID THINKING)
       (GOAL ISA GOAL STATE START-ACTION [SRC-TARGET] EMPTY [DEST-TARGET] EMPTY %LAST-LOCATION EMPTY)
     )
       (P OBJECT-A-NOT-FOUND
         =GOAL>
           ISA             GOAL
           STATE             ENCODE-OBJECT-A-ACTION
         =VISUAL>
           ISA             TEXT
         - VALUE             "object1"
         =VISUAL-LOCATION>
           ISA             VISUAL-LOCATION
       ==>
         =GOAL>
           STATE             FIND-OBJECT-A-ACTION
       )
       (P THINKING-TASK
         =GOAL>
           ISA             GOAL
           STATE             THINKING
       ==>
         =GOAL>
           STATE             FIND-OBJECT-A-ACTION
       )
       (P FIND-OBJECT-A
         =GOAL>
           ISA             GOAL
           STATE             FIND-OBJECT-A-ACTION
         ?visual>
           STATE                   FREE
       ==>
         =GOAL>
           STATE             ATTEND-OBJECT-A-ACTION

         +VISUAL-LOCATION>
           ISA             a-object-location
           :attended                NIL
       )
       (P ATTEND-OBJECT-A
         =GOAL>
           ISA             GOAL
           STATE             ATTEND-OBJECT-A-ACTION

         =VISUAL-LOCATION>
           ISA             a-object-location
           SCREEN-X           =X
         ?visual>
           STATE             FREE
       ==>
         =GOAL>
           STATE             ENCODE-OBJECT-A-ACTION
         +VISUAL>
           ISA             MOVE-ATTENTION
           SCREEN-POS          =VISUAL-LOCATION
         +VISUAL-LOCATION>
           ISA             a-object-location
       )
       (P ENCODE-OBJECT-A
         =GOAL>
           ISA             GOAL
           STATE             ENCODE-OBJECT-A-ACTION
         =VISUAL>
           ISA             a-object
           VALUE               =TEXT
           VALUE             "object1"
         =VISUAL-LOCATION>
           ISA             a-object-location
```

```
==>
  =GOAL>
    [SRC-TARGET]          =VISUAL-LOCATION
    STATE                 HAND-TO-MOUSE-FOR-A-ACTION
 +VISUAL-LOCATION>
    ISA                   VISUAL-LOCATION
)
(P HAND-TO-MOUSE-FOR-A
  =GOAL>
    ISA                   GOAL
    STATE                 HAND-TO-MOUSE-FOR-A-ACTION
  ?manual>
    STATE                 FREE
==>
  =GOAL>
    STATE                 MOUSE-MOVES-A-ACTIONS
  +MANUAL>
    ISA                   HAND-TO-MOUSE
)
(P MOVE-MOUSE-TO-A
  =GOAL>
    ISA                   GOAL
    [SRC-TARGET]          =THETARGET-A
    STATE                 MOUSE-MOVES-A-ACTIONS
  ?manual>
    STATE                 FREE
==>
  =GOAL>
    %LAST-LOCATION        =THETARGET-A
    [SRC-TARGET]          EMPTY
    STATE                 CLICK-MOUSE-OVER-A-ACTION
  +MANUAL>
    ISA                   MOVE-CURSOR
    LOC                    =THETARGET-A
)
(P CLICK-MOUSE-OVER-A
  =GOAL>
    ISA                   GOAL
    STATE                 CLICK-MOUSE-OVER-A-ACTION
  ?manual>
    STATE                 FREE
==>
  =GOAL>
    STATE                 FIND-OBJECT-B-ACTION
  +MANUAL>
    ISA                   CLICK-MOUSE
)
(P OBJECT-B-NOT-FOUND
  =GOAL>
    ISA                   GOAL
    STATE                 ENCODE-OBJECT-B-ACTION
  =VISUAL>
    ISA                   TEXT
   - VALUE                "object2"
  =VISUAL-LOCATION>
    ISA                   VISUAL-LOCATION
==>
  =GOAL>
    STATE                 FIND-OBJECT-B-ACTION
)
(P FIND-OBJECT-B
  =GOAL>
    ISA                   GOAL
    STATE                 FIND-OBJECT-B-ACTION
  ?visual>
    STATE                       FREE
==>
  =GOAL>
    STATE                 ATTEND-OBJECT-B-ACTION

  +VISUAL-LOCATION>
    ISA                   b-object-location
    :ATTENDED                   NIL
)
(P ATTEND-OBJECT-B
  =GOAL>
    ISA                   GOAL
    STATE                 ATTEND-OBJECT-B-ACTION
  =VISUAL-LOCATION>
    ISA                   b-object-location
==>
  =GOAL>
    STATE                 ENCODE-OBJECT-B-ACTION
  +VISUAL>
     ISA                  MOVE-ATTENTION
    SCREEN-POS            =VISUAL-LOCATION
  +VISUAL-LOCATION>
    ISA                   b-object-location
)
(P ENCODE-OBJECT-B
  =GOAL>
    ISA                   GOAL
    STATE                 ENCODE-OBJECT-B-ACTION
  =VISUAL>
    ISA                   b-object
    VALUE                 =TEXT
    VALUE                 "object2"
  =VISUAL-LOCATION>
    ISA                   VISUAL-LOCATION
```

```
    ==>
      =GOAL>
        [DEST-TARGET]         =VISUAL-LOCATION
        STATE                 HAND-TO-MOUSE-FOR-B-ACTION
    )
    (P HAND-TO-MOUSE-FOR-B
      =GOAL>
        ISA                   GOAL
        STATE                 HAND-TO-MOUSE-FOR-B-ACTION
      ?manual>
        STATE                 FREE
    ==>
      =GOAL>
        STATE                 MOVE-MOUSE-TO-B-ACTION
      +MANUAL>
        ISA                   HAND-TO-MOUSE
    )
    (P MOVE-MOUSE-TO-B
      =GOAL>
        ISA                   GOAL
        [DEST-TARGET]         =THETARGET-B
        STATE                 MOVE-MOUSE-TO-B-ACTION
      ?manual>
        STATE                 FREE
    ==>
      =GOAL>
        %LAST-LOCATION        =THETARGET-B
        [DEST-TARGET]         EMPTY
        STATE                 MOUSE-MOVES-B-ACTIONS

      +MANUAL>
        ISA                   MOVE-CURSOR
        LOC                    =THETARGET-B
    )
    (P CLICK-MOUSE-OVER-B
      =GOAL>
        ISA                   GOAL
        STATE                 MOUSE-MOVES-B-ACTIONS
      ?manual>
        STATE                 FREE
    ==>
      +MANUAL>
      ISA                   CLICK-MOUSE
      -GOAL>
    )
    (SPP THINKING-TASK :at 1.2)
    (GOAL-FOCUS GOAL)
    (sgp :v T :needs-mouse T :show-focus t :trace-detail high)
)
(cond
      (
         (> ,thinking 0)
         (p-fct
             (list
               'START-TASK
                 '=GOAL>
                   'ISA              'GOAL
                   'STATE            'START-ACTION
                '==>

                 '=GOAL>
                   'STATE            'THINKING
             )
         )
         (SPP THINKING-TASK :at 1.2)
      )
      (
         (= ,thinking 0)
         (p-fct
               (list
               'START-TASK
                 '=GOAL>
                   'ISA              'GOAL
                   'STATE            'START-ACTION
               '==>
                 '=GOAL>
                   'STATE            'FIND-OBJECT-A-ACTION
               )
          )
      )
)
(let (
      (visual-location-chunks nil)
      (visual-object-chunks  nil)
    )
  (push (car (define-chunks-fct `((
      VISUAL-SW1-OBJECT
      isa a-object-location
      color      black
      value      a-object
      kind       a-object
      screen-x   ,obj1X
      screen-y   ,obj1Y
      width      ,obj1W
      height     ,obj1H)))) visual-location-chunks )
  (push (car (define-chunks-fct `((
      VISUAL-SW2-OBJECT
      isa b-object-location
      color      black
```

```
            value      b-object
            kind       b-object
            screen-x   ,obj2X
            screen-y   ,obj2Y
            width      ,obj2W
            height     ,obj2H)))) visual-location-chunks )
        (push (define-chunks-fct `((
            SW1-OBJECT
            isa        a-object
            CONTENT    "object1"
            value      "object1"
            width      ,obj1W
            height     ,obj1H))) visual-object-chunks )
        (push (define-chunks-fct `((
            SW2-OBJECT
            isa        b-object
            CONTENT    "object2"
            value      "object2"
            width      ,obj2W
            height     ,obj2H))) visual-object-chunks )
        (let (
              (the-device (pairlis visual-location-chunks visual-object-chunks))
            )
            (install-device the-device)
            (proc-display)
            (run 100)
            (delete-model)
        )
      )
    )
  )
```

# APPENDIX D: SAMPLE RUN USING THE DRAG AND DROP MACRO

Run for: drag-drop-Action - 55 - 7 - 10 - 40 | 267 - 237 - 50 - 40 | 1

```
0.000  GOAL            SET-BUFFER-CHUNK GOAL GOAL REQUESTED NIL
0.000  BUFFER            Buffer GOAL copied chunk GOAL to GOAL-0
0.000  VISION          SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0 REQUESTED NIL
0.000  BUFFER            Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-0
0.000  PROCEDURAL            CONFLICT-RESOLUTION
0.000  PROCEDURAL            PRODUCTION-SELECTED START-TASK
0.000  PROCEDURAL            BUFFER-READ-ACTION GOAL
0.050  PROCEDURAL            PRODUCTION-FIRED START-TASK
0.050  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
0.050  PROCEDURAL            CONFLICT-RESOLUTION
0.050  PROCEDURAL            PRODUCTION-SELECTED THINKING-TASK
0.050  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.250  PROCEDURAL            PRODUCTION-FIRED THINKING-TASK
1.250  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
1.250  PROCEDURAL            CONFLICT-RESOLUTION
1.250  PROCEDURAL            PRODUCTION-SELECTED FIND-OBJECT-A
1.250  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.250  PROCEDURAL            QUERY-BUFFER-ACTION VISUAL
1.300  PROCEDURAL            PRODUCTION-FIRED FIND-OBJECT-A
1.300  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
1.300  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
1.300  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
1.300  VISION              Find-location
1.300  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
1.300  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-1
1.300  PROCEDURAL            CONFLICT-RESOLUTION
1.300  PROCEDURAL            PRODUCTION-SELECTED ATTEND-OBJECT-A
1.300  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.300  PROCEDURAL            BUFFER-READ-ACTION VISUAL-LOCATION
1.300  PROCEDURAL            QUERY-BUFFER-ACTION VISUAL
1.350  PROCEDURAL            PRODUCTION-FIRED ATTEND-OBJECT-A
1.350  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
1.350  PROCEDURAL            MODULE-REQUEST VISUAL
1.350  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
1.350  PROCEDURAL            CLEAR-BUFFER VISUAL
1.350  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
1.350  VISION              Move-attention VISUAL-SW1-OBJECT-0-1 NIL
1.350  VISION              Find-location
1.350  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
1.350  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-2
1.350  PROCEDURAL            CONFLICT-RESOLUTION
1.500  VISION              Preparation-complete TRGT0
1.500  PROCEDURAL            CONFLICT-RESOLUTION
1.576  VISION              Complete-eye-movement TRGT0 #(58 8)
1.576  PROCEDURAL            CONFLICT-RESOLUTION
1.617  VISION              Encoding-Complete VISUAL-SW1-OBJECT-0-1
1.617  VISION              SET-BUFFER-CHUNK VISUAL SW1-OBJECT
1.617  BUFFER              Buffer VISUAL copied chunk SW1-OBJECT to SW1-OBJECT-0
1.617  PROCEDURAL            CONFLICT-RESOLUTION
1.617  PROCEDURAL            PRODUCTION-SELECTED ENCODE-OBJECT-A
1.617  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.617  PROCEDURAL            BUFFER-READ-ACTION VISUAL
1.617  PROCEDURAL            BUFFER-READ-ACTION VISUAL-LOCATION
1.667  PROCEDURAL            PRODUCTION-FIRED ENCODE-OBJECT-A
1.667  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
1.667  PROCEDURAL            MODULE-REQUEST VISUAL-LOCATION
1.667  PROCEDURAL            CLEAR-BUFFER VISUAL
1.667  PROCEDURAL            CLEAR-BUFFER VISUAL-LOCATION
1.667  VISION              Find-location
1.667  VISION              SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW1-OBJECT-0
1.667  BUFFER              Buffer VISUAL-LOCATION copied chunk VISUAL-SW1-OBJECT-0 to VISUAL-SW1-OBJECT-0-3
1.667  PROCEDURAL            CONFLICT-RESOLUTION
1.667  PROCEDURAL            PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-A
1.667  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.667  PROCEDURAL            QUERY-BUFFER-ACTION MANUAL
1.676  VISION              Preparation-complete TRGT0
1.717  PROCEDURAL            PRODUCTION-FIRED HAND-TO-MOUSE-FOR-A
1.717  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
1.717  PROCEDURAL            MODULE-REQUEST MANUAL
1.717  PROCEDURAL            CLEAR-BUFFER MANUAL
1.717  MOTOR               HAND-TO-MOUSE
1.717  PROCEDURAL            CONFLICT-RESOLUTION
1.717  PROCEDURAL            PRODUCTION-SELECTED MOVE-MOUSE-TO-A
1.717  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.717  PROCEDURAL            QUERY-BUFFER-ACTION MANUAL
1.746  VISION              Complete-eye-movement TRGT0 #(55 7)
1.767  PROCEDURAL            PRODUCTION-FIRED MOVE-MOUSE-TO-A
1.767  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
1.767  PROCEDURAL            MODULE-REQUEST MANUAL
1.767  PROCEDURAL            CLEAR-BUFFER MANUAL
1.767  MOTOR               MOVE-CURSOR OBJECT NIL LOC VISUAL-SW1-OBJECT-0-2
1.767  PROCEDURAL            CONFLICT-RESOLUTION
1.767  PROCEDURAL            PRODUCTION-SELECTED CLICK-MOUSE-OVER-A
1.767  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.767  PROCEDURAL            QUERY-BUFFER-ACTION MANUAL
1.817  PROCEDURAL            PRODUCTION-FIRED CLICK-MOUSE-OVER-A
1.817  PROCEDURAL            MOD-BUFFER-CHUNK GOAL
1.817  PROCEDURAL            MODULE-REQUEST MANUAL
1.817  PROCEDURAL            CLEAR-BUFFER MANUAL
1.817  MOTOR               CLICK-MOUSE
1.817  PROCEDURAL            CONFLICT-RESOLUTION
1.817  PROCEDURAL            PRODUCTION-SELECTED FIND-OBJECT-B
1.817  PROCEDURAL            BUFFER-READ-ACTION GOAL
1.817  PROCEDURAL            QUERY-BUFFER-ACTION VISUAL
```

```
1.867  PROCEDURAL        PRODUCTION-FIRED FIND-OBJECT-B
1.867  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
1.867  PROCEDURAL        MODULE-REQUEST VISUAL-LOCATION
1.867  PROCEDURAL        CLEAR-BUFFER VISUAL-LOCATION
1.867  VISION            Find-location
1.867  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
1.867  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-0
1.867  PROCEDURAL        CONFLICT-RESOLUTION
1.867  PROCEDURAL        PRODUCTION-SELECTED ATTEND-OBJECT-B
1.867  PROCEDURAL        BUFFER-READ-ACTION GOAL
1.867  PROCEDURAL        BUFFER-READ-ACTION VISUAL-LOCATION
1.917  PROCEDURAL        PRODUCTION-FIRED ATTEND-OBJECT-B
1.917  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
1.917  PROCEDURAL        MODULE-REQUEST VISUAL
1.917  PROCEDURAL        MODULE-REQUEST VISUAL-LOCATION
1.917  PROCEDURAL        CLEAR-BUFFER VISUAL
1.917  PROCEDURAL        CLEAR-BUFFER VISUAL-LOCATION
1.917  VISION            Move-attention VISUAL-SW2-OBJECT-0-0 NIL
1.917  VISION            Find-location
1.917  VISION            SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-SW2-OBJECT-0
1.917  BUFFER             Buffer VISUAL-LOCATION copied chunk VISUAL-SW2-OBJECT-0 to VISUAL-SW2-OBJECT-0-1
1.917  PROCEDURAL        CONFLICT-RESOLUTION
1.967  MOTOR             PREPARATION-COMPLETE
1.967  PROCEDURAL        CONFLICT-RESOLUTION
2.017  VISION            Preparation-complete TRGT1
2.017  MOTOR             INITIATION-COMPLETE
2.017  PROCEDURAL        CONFLICT-RESOLUTION
2.027  MOTOR             OUTPUT-KEY #(28 2)
2.027  PROCEDURAL        CONFLICT-RESOLUTION
2.117  MOTOR             FINISH-MOVEMENT
2.117  PROCEDURAL        CONFLICT-RESOLUTION
2.119  VISION            Complete-eye-movement TRGT1 #(268 233)
2.119  PROCEDURAL        CONFLICT-RESOLUTION
2.177  VISION            Encoding-Complete VISUAL-SW2-OBJECT-0-0
2.177  VISION            SET-BUFFER-CHUNK VISUAL SW2-OBJECT
2.177  BUFFER             Buffer VISUAL copied chunk SW2-OBJECT to SW2-OBJECT-0
2.177  PROCEDURAL        CONFLICT-RESOLUTION
2.177  PROCEDURAL        PRODUCTION-SELECTED ENCODE-OBJECT-B
2.177  PROCEDURAL        BUFFER-READ-ACTION GOAL
2.177  PROCEDURAL        BUFFER-READ-ACTION VISUAL
2.177  PROCEDURAL        BUFFER-READ-ACTION VISUAL-LOCATION
2.219  VISION            Preparation-complete TRGT1
2.227  PROCEDURAL        PRODUCTION-FIRED ENCODE-OBJECT-B
2.227  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
2.227  PROCEDURAL        CLEAR-BUFFER VISUAL
2.227  PROCEDURAL        CLEAR-BUFFER VISUAL-LOCATION
2.227  PROCEDURAL        CONFLICT-RESOLUTION
2.227  PROCEDURAL        PRODUCTION-SELECTED HAND-TO-MOUSE-FOR-B
2.227  PROCEDURAL        BUFFER-READ-ACTION GOAL
2.227  PROCEDURAL        QUERY-BUFFER-ACTION MANUAL
2.277  PROCEDURAL        PRODUCTION-FIRED HAND-TO-MOUSE-FOR-B
2.277  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
2.277  PROCEDURAL        MODULE-REQUEST MANUAL
2.277  PROCEDURAL        CLEAR-BUFFER MANUAL
2.277  MOTOR             HAND-TO-MOUSE
2.277  PROCEDURAL        CONFLICT-RESOLUTION
2.277  PROCEDURAL        PRODUCTION-SELECTED MOVE-MOUSE-TO-B
2.277  PROCEDURAL        BUFFER-READ-ACTION GOAL
2.277  PROCEDURAL        QUERY-BUFFER-ACTION MANUAL
2.289  VISION            Complete-eye-movement TRGT1 #(267 237)
2.327  PROCEDURAL        PRODUCTION-FIRED MOVE-MOUSE-TO-B
2.327  PROCEDURAL        MOD-BUFFER-CHUNK GOAL
2.327  PROCEDURAL        MODULE-REQUEST MANUAL
2.327  PROCEDURAL        CLEAR-BUFFER MANUAL
2.327  MOTOR             MOVE-CURSOR OBJECT NIL LOC VISUAL-SW2-OBJECT-0-1
2.327  PROCEDURAL        CONFLICT-RESOLUTION
2.527  MOTOR             PREPARATION-COMPLETE
2.527  PROCEDURAL        CONFLICT-RESOLUTION
2.577  MOTOR             INITIATION-COMPLETE
2.577  PROCEDURAL        CONFLICT-RESOLUTION
2.838  MOTOR             MOVE-CURSOR-ABSOLUTE #(267 237)
2.838  PROCEDURAL        CONFLICT-RESOLUTION
2.888  MOTOR             FINISH-MOVEMENT
2.888  PROCEDURAL        CONFLICT-RESOLUTION
2.888  PROCEDURAL        PRODUCTION-SELECTED CLICK-MOUSE-OVER-B
2.888  PROCEDURAL        BUFFER-READ-ACTION GOAL
2.888  PROCEDURAL        QUERY-BUFFER-ACTION MANUAL
2.938  PROCEDURAL        PRODUCTION-FIRED CLICK-MOUSE-OVER-B
2.938  PROCEDURAL        MODULE-REQUEST MANUAL
2.938  PROCEDURAL        CLEAR-BUFFER GOAL
2.938  PROCEDURAL        CLEAR-BUFFER MANUAL
2.938  MOTOR             CLICK-MOUSE
2.938  PROCEDURAL        CONFLICT-RESOLUTION
3.088  MOTOR             PREPARATION-COMPLETE
3.088  PROCEDURAL        CONFLICT-RESOLUTION
3.138  MOTOR             INITIATION-COMPLETE
3.138  PROCEDURAL        CONFLICT-RESOLUTION
3.148  MOTOR             OUTPUT-KEY #(28 2)
3.148  PROCEDURAL        CONFLICT-RESOLUTION
3.238  MOTOR             FINISH-MOVEMENT
3.238  PROCEDURAL        CONFLICT-RESOLUTION
3.238  ------            Stopped because no events left to process

Execution time for drag-drop-Action is 3.238
```

# APPENDIX E: LISP HELPER DEBUGGING FUNCTIONALITY CODE

Stepper Code

```
(defun start-threaded-process (name function)
  (setf *threaded-process* (mp::process-run-function name #'(lambda ()(in-package :cl-user)(funcall function))))
)


(defun threaded-process-kill (process)
 (if
   (not(equal process nil))
    (mp::process-kill process)
   )
)

(defun stop-threaded-process()
 (if
  (not (equal *threaded-process* nil))
  (progn
   (format *error-output* "Stopping the threaded process. threadRunning:~%")
   (threaded-process-kill *threaded-process*)
   (setf *threaded-process* nil)
  )
 )
)

(defvar *stepper-process* nil)
(defvar *stepper-thread-running* nil)


(defun stepper-run-process (name function)
  (setf *stepper-process* (mp::process-run-function name #'(lambda ()(in-package :cl-user)(funcall function))))
)

(defun stepper-process-kill (process)
 (if
   (not(equal process nil))
    (mp::process-kill process)
   )
)

(defun stepper-process-loop()
 (if
   (not(equal *stepper-process* nil))
    (progn
     (if
      (not(equal *stepper-thread-running* t))
       (progn
        (setf *stepper-thread-running* t)
        (run-my-stepper)
        (sleep 2)
       )
     )
    )
 )
)

(defun stop-stepper-process()
 (if
  (not (equal *stepper-process* nil))
  (progn
   (stepper-process-kill *stepper-process*)
   (setf *stepper-process* nil)
  )
 )
)

(defun start-stepper-process()
 (stepper-run-process "Running Lisp stepper application" #'(lambda () (run-my-stepper)))
)

(defun updateDLLStepperInfo(info)
 ( updateSteppingInformation info)
)

(defun run-my-stepper ()
 (stepperRunStartedToNative)

 (setf *abort-stepping* nil)
 (verify-current-mp
  "run-step called with no current meta-process."
  (flet (
         (test (mp next-time count)
            (declare (ignore next-time count))
            (loop
               (let (
                   (event (car (meta-p-events mp)))
                )

                (when (null (meta-p-events mp))
                   (format t "No more events to process")
                   (stepperRunFinishedToNative)
                   (return t)
                 )
                ( updateSteppingInformation
                 (format nil "~A|~A|~A|~A|~A|~A~%"
```

```
                (evt-time event)
                (evt-module event)
                (evt-details event)
                (evt-action event)
                (evt-destination event)
                (evt-params event)

               )
              )
                (while *wait-for-stepping*
                  (uni-process-system-events)
                  (sleep 0.1)
                  (if
                   (equal *abort-stepping* t)
                   (progn
                     (format t "~%------STEPPER STOPPING --------~%" )
                     (stepperRunFinishedToNative)
                     (return t)
                    )
                   )
                  )
                 (if
                   (not (equal *abort-stepping* t))
                   (progn
                    (setf *wait-for-stepping* T)
                    (return nil)
                   )
                   (progn
                    (format t "~%------STEPPER STOPPED --------~%" )
                    (stepperRunFinishedToNative)
                    (return t)
                   )
                 )
               )
          )
         )
        )
    (multiple-value-bind (time events break)
     (run-sched-queue (current-mp) #'test :real-time nil)
     (unless break
      (send-run-terminated-events (current-mp))
      (stepperRunFinishedToNative)
     )
     (values time events break)
   )
  )
 )
)
(defun stepperStep ()
(has-current-mp
 "No Meta process present."
 (has-current-model
 "No model loaded"
 (setf *wait-for-stepping* nil)
 )
 )
)

(defun initialisingStepper ()
(has-current-mp
 "No Meta process present."
 (has-current-model
 "No model loaded"
 (start-stepper-process)
 )
 )
)

(defun removingStepper ()
(has-current-mp
 "No Meta process present."
 (has-current-model
 "No model loaded"
 (setf *abort-stepping* t)
 (stop-stepper-process)
 )
 )
)

Procedural rule information retrieval

(defun return-productions ()
(has-current-mp
"No Meta process present."
(has-current-model
 "No model loaded"

 (let ((prod (get-module procedural)))
   (if prod
      (procedural-productions prod)
      (print-warning "No procedural module found")
   )
 )
 )
 )
)

(defun return-production-name-cs(cs)
```

```
 (production-name  (get-production (car cs))  )
)

(defun return-production-name(prod)
  (production-name  prod)
)

(defun production-u-value (prod)
   (caar (no-output (spp-fct (list prod :u)))))


(defun return-productions-names ()
          (has-current-mp
          "No Meta process present."
          (has-current-model
           "No model loaded"
            (let ((prod (get-module procedural)))
              (if prod
                 (let (
                       (names-list nil)
                       (all-productions-names  (mapcar #'production-name (procedural-productions prod))  )
                   )
                  (dolist (production-name all-productions-names)
                    (let ((production (get-production production-name)))
                      (if production
                         (progn
                        (push "|" names-list)
                           (push production-name names-list)
                         )
                         (format nil "No production named ~S is defined" production-name)
                       )
                     )
                    )
                    (reverse names-list)
                  )
                 (format nil "No procedural module found")
               )
             )
            )
     )
)

(defun send-DLL-productions-names()
          (has-current-mp
                    "No Meta process present."
                    (has-current-model
                         "No model loaded"
                         (format nil "~a" (return-productions-names) )
                    )
          )
)

(defun send-DLL-productions-parameters(prod-name)
  (has-current-mp
  "No Meta process present."
  (has-current-model
    "No model loaded"
     (getProductionUtilities (get-Production-Parameters-From-Name prod-name))
   )
  )
)

(defun coerce-our-list-to-vector( mylist)
  (let (
        (myinternallist  (coerce  mylist  'vector ))
     )
     myinternallist
  )
)

(defun coerce-a-list-to-vector-pairs( mylist)
  (let (
     (myinternallist  (coerce  mylist  'vector ))
     (count 3)
     (my-vector nil)
   )
   (setq  my-vector  (make-array  '(count 2)))
  )
)

(defun get-Production-Information-From-Name(prodname)
  (formatOutput-Production-Details-From-Name prodname)
)


Chunks information retrieval

(defun return-chunks-names ()
  (has-current-mp
  "No Meta process present."
    (has-current-model
    "No model loaded"
      (let ((dm (get-module declarative)))
        (if dm
          (let (
           (names-list nil)
           )
           (dolist (chunk (all-dm-chunks dm))
            (if chunk
```

```
            (progn
              (push "|" names-list)
              (push chunk names-list)
            )
            (format nil "Declarative chunk not defined")
          )
        )
        (reverse names-list)
      )
      (format nil "No declarative module found")
    )
   )
  )
 )
)

(defun send-DLL-chunks-names()
 (has-current-mp
 "No Meta process present."
  (has-current-model
   "No model loaded"
    (format nil "~a" (return-chunks-names ) )
  )
 )
)

(defun display-all-chunks ()
 (has-current-mp
 "No Meta process present."
  (has-current-model
  "No model loaded"
   (let ((dm (get-module declarative)))
    (if dm
     (dolist (chunk (all-dm-chunks dm))
      (format nil "~%CHUNK IS: ~a" chunk)
     )
     (format nil "No declarative memory module found")
    )
   )
  )
 )
)

(defconstant *JCG-pprint-chunk-string*
  (formatter
   "~S  ~:[ (~s)~;~*~] ~%~@[~S~%~]   ISA ~S~%  ************| ~:{   ~s  ~s~%~} |**********")
   "compiled format string for printing chunks")
)


(defun JCG-pprint-a-chunk (chunk-name &optional (w-params t))
 (has-current-mp
 "No Meta process present."
  (has-current-model
  "No model loaded"
  (let (
    (chunk (get-chunk chunk-name))
    (resOut "")
    )
    (if chunk
     (progn
      (setf resOut
       (format nil
        "~S~:[ (~s)~;~*~]~%~@[~S~%~]  ISA ~S~%~:{   ~s  ~s~%~} ~%  ~@[--Chunk Parameters--~%~:{   ~s  ~s~%~}~]~%~%"
        chunk-name
        (eql chunk-name (act-r-chunk-name chunk))
        (act-r-chunk-name chunk)
        (act-r-chunk-documentation chunk)
        (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))
        (mapcan #'(lambda (slot-name)
          (multiple-value-bind (value exists) (gethash slot-name (act-r-chunk-slot-value-lists chunk))
            (when (or exists
              (car (no-output (sgp-fct (list :show-all-slots))))
              (not (extended-slot-name-p slot-name (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))))
              )
              (list (list slot-name value))
              )
            )
          )
          (ct-slot-names (act-r-chunk-chunk-type chunk))
        )
        (mapcar #'(lambda (param)
           (list (act-r-chunk-parameter-name param)
            (progn
             (funcall (act-r-chunk-parameter-accessor param) chunk-name)
            )
           )
          )
          *chunk-parameters-list*
        )
       )
      )
      resOut
     )
     (format nil "~%No chunk of name:~a~% " chunk-name)
    )
   )
  )
 )
)
```

```
)
```

Loading ACT-R models

```
 (defun loadingModel(filename)
   (declare (ignore filename))
  (if (or *stepper-open* *running-actr*)
    (list 0 "ACT-R currently running or the stepper is open.")
    (unwind-protect
     (progn
      (setf *running-actr* t)
      (let ((result (safe-load filename t)))
       (setf *running-actr* nil)
       (format t (if (= 1 (first result)) "~%#|## Model ~a loaded. ##|#~%" "~%#|## Failed loading model ~a. ##|#~%") filename)
      result
      )
     )
    (setf *running-actr* nil)
    )
 )
)


(defun runningModel (run_time)
 (declare (ignore run_time))
 (unwind-protect
   (if *running-actr*
    (model-warning "Run button has no effect because model still running when pressed.")
    (progn (setf *running-actr* t)
     (run run_time)
    )
   )
   (setf *running-actr* nil)
 )
)


(defun reloadingModel ()
 (has-current-mp
 "No Meta process present."
  (has-current-model
   "No model loaded"
    (reload-model nil)
  )
 )
)
```

Buffers information retrieval

```
(defun JCG-buffer-status-fct (buffer-names-list)
 (has-current-mp
 "No Meta process present."
  (has-current-model
  "No model loaded"
  (let (
     (res nil)
     (resOut "")
    )
    (dolist (buffer-name (if buffer-names-list
       buffer-names-list
       (buffers))
       res)
     (let ((buffer (buffer-instance buffer-name)))
      (if buffer
       (progn
        (setf resOut
         (format nil
         "Buffer Name:  ~S |
         Buffer Empty:  ~S |
         Buffer Full:  ~S |
         Buffer Requested:  ~S |
         Buffer Unrequested:  ~S |
         State Free:  ~S |
         State Error:  ~S |
         "
         buffer-name
         (query-buffer buffer-name '((buffer . empty)))
         (query-buffer buffer-name '((buffer . full)))
         (query-buffer buffer-name '((buffer . requested)))
         (query-buffer buffer-name '((buffer . unrequested)))
         (query-buffer buffer-name '((state . free)))
         (query-buffer buffer-name '((state . error)))
         )
        )
       )
      )
     )
    )
    resOut
  )
  )
 )
)

(defun JCG-buffer-chunk-fct (buffer-names-list)
 (has-current-mp
 "No Meta process present."
  (has-current-model
```

```
   "No model loaded"
    (let (
     (res nil)
     (resOut "")
    )
    (dolist (buffer-name (if buffer-names-list
       buffer-names-list
       (buffers))
       res)
     (let* ((buffer (buffer-instance buffer-name)))
      (if buffer
       (let ((chunk (act-r-buffer-chunk buffer)))
        (setf resOut
         (format nil "Buffer Name: ~S ~%Chunk Inside: ~S ~%Base Chunk: ~@[[~s]~] ~%~%--Chunk Information--~%~a~%"
          buffer-name
          chunk
          (when chunk
           (chunk-copied-from-fct chunk)
          )
          (when buffer-names-list
           (progn
            (JCG-pprint-a-chunk chunk)
           )
          )
         )
        )
        (push-last (if buffer-names-list
         chunk
         (cons buffer-name chunk))
         res)
       )
       (push-last (if buffer-names-list
       :error
        (cons :error nil))
       res)
      )
     )
    )
    resOut
    )
   )
  )
)

(defun return-buffers-names ()
  (has-current-mp
  "No Meta process present."
   (has-current-model
   "No model loaded"
   (let (
    (names-list nil)
    )
    (dolist (buffer (buffers))
     (if buffer
      (progn
       (push "|" names-list)
       (push buffer names-list)
      )
      (format nil "Buffer not defined")
     )
    )
    (reverse names-list)
   )
   )
  )
)


(defun send-DLL-buffers-names()
  (has-current-mp
  "No Meta process present."
   (has-current-model
   "No model loaded"
    (format nil "~a" (return-buffers-names ) )
   )
  )
)

(defun get-Chunk-Parameters-From-Name(chunk)
  (has-current-mp
  "No Meta process present."
   (has-current-model
   "No model loaded"
    (get-Chunk-Information-From-Name chunk)
   )
  )
)

(defun get-Buffer-Parameters-From-Name(buffer)
(has-current-mp
  "No Meta process present."
  (has-current-model
 "No model loaded"
  (get-Buffer-Information-From-Name buffer)
   )
   )
)
```

```
(defun get-BufferStates-Parameters-From-Name(buffer)
 (has-current-mp
 "No Meta process present."
  (has-current-model
  "No model loaded"
   (format nil "~a"  (JCG-buffer-status-fct (list (intern buffer))))
  )
 )
)

(defun get-Buffer-Information-From-Name(buffer-name)
 (has-current-mp
 "No Meta process present."
  (has-current-model
  "No model loaded"
   (format nil "~a" (JCG-buffer-chunk-fct (list (intern buffer-name))))
  )
 )
)
(defun get-BufferStates-Information-From-Name(buffer-name)
 (has-current-mp
 "No Meta process present."
  (has-current-model
  "No model loaded"
   (format nil "~a" (JCG-buffer-status-fct (list (intern buffer-name))))
  )
 )
)

(defun get-Chunk-Information-From-Name(chunk-name)
 (has-current-mp
 "No Meta process present."
  (has-current-model
  "No model loaded"
  (let ((dm (get-module declarative)))
  (if dm
  (let* (
     (chunk (intern chunk-name))
     (similarChunks (get-all-chunk-similarities dm chunk))
     (thechunk (get-chunk chunk))
     (aString (string (format nil "~%[~a] buffer Chunk Details: " chunk-name)))
     )
    (if chunk
     (progn
      (setf aString (concatenate 'string aString
       (format nil "~%Details for chunk ~S inside ~a buffer: ~:[ (~s) ~; ~* ~] ~% ~@[~S~%~]  ISA ~S ~% ~:{   ~s  ~s~%~}"
        chunk-name
        (act-r-chunk-name thechunk)
        (eql theChunk (act-r-chunk-name thechunk))
        (act-r-chunk-name thechunk)
        (act-r-chunk-documentation thechunk)
        (act-r-chunk-type-name (act-r-chunk-chunk-type thechunk))
        (mapcan #'(lambda (slot-name)
        (multiple-value-bind (value exists) (gethash slot-name (act-r-chunk-slot-value-lists thechunk))
       (when (or exists
        (car (no-output (sgp-fct (list :show-all-slots))))
        (not
         (extended-slot-name-p slot-name (act-r-chunk-type-name (act-r-chunk-chunk-type thechunk))   )
        )
        )
        (list (list slot-name value))
       )))
       (ct-slot-names (act-r-chunk-chunk-type thechunk))
       )))
      )
      (setf aString (concatenate 'string aString

      (format nil "~@[~%parameters for chunk ~a inside buffer:~a~%~:{   ~s  ~s~%~}~]~%"
       chunk-name
       (act-r-chunk-name theChunk)
       (mapcar #'(lambda (param)
          (list (act-r-chunk-parameter-name param)
           (funcall (act-r-chunk-parameter-accessor param) chunk)
          )
         )
         *chunk-parameters-list*
      ))))
      aString
     )
     (format nil "Not a chunk- NIL")
    )
   )
   (format nil "No declarative module found")
  ))))
)

IDTR Hook helper code for information retrieval

(defun IDTR-chunk-merge-hook (mc)
)

(defun formatOutput-chunk-spec (chunk-spec)
 (let (
      (aString (string ""))
    )
   (when (act-r-chunk-spec-p chunk-spec)
    (setf aString (concatenate 'string aString  (format nil "ISA ~A" (chunk-spec-chunk-type chunk-spec))))
    (dolist (slot (act-r-chunk-spec-slots chunk-spec))
     (if (eql '= (act-r-slot-spec-modifier slot))
```

```
        (setf aString (concatenate 'string aString (format nil "~%~s ~S"
           (act-r-slot-spec-name slot)
           (act-r-slot-spec-value slot)  ))
        )
        (setf aString (concatenate 'string aString (format nil "~% ~2a ~s ~S"
           (act-r-slot-spec-modifier slot)
           (act-r-slot-spec-name slot)
           (act-r-slot-spec-value slot)  ))
        )
      )
     )
    )
   aString
  )
)

(defun IDTR-retrieval-request-hook (request)
  (if (not (null request))
    (let* (
          (request-type (act-r-chunk-spec-type request))
          (request-type-name (act-r-chunk-type-name request-type))
          (request-type-slots (act-r-chunk-type-slots request-type))
         )
        (hookOutputInformation 0 "retrieval-request" "Chunk" (coerce
        (format nil  "Request type name:       ~a
           ~%request slots:~%~a
           ~%request bindings:~%~a
           ~%request:~%~a~%
           "
           request-type-name
           request-type-slots
           (formatOutput-chunk-spec request)
           request
        )
        'string ) 1)
      )
 )
)

(defun IDTR-retrieval-set-hook (ret-set)
   (dolist (x ret-set)
     (if x
       (let* (
            (chunk (get-chunk x))
            (theChunk (act-r-buffer-chunk chunk))
            (chunk-name (act-r-chunk-name chunk))
            (aString (string (format nil "Buffer: ~a" chunk-name)))
           )
       (if chunk
         (progn
          (setf aString (concatenate 'string aString
           (format nil "~%~%Details for chunk ~a: ~:[ (~s) ~; ~* ~] ~% ~@[~S~%~]  ISA ~S ~% ~:{   ~s  ~s~%~}"
                   chunk-name
                  (eql chunk-name (act-r-chunk-name chunk))
                  (act-r-chunk-name chunk)
                  (act-r-chunk-documentation chunk)
                  (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))
                  (mapcan #'(lambda (slot-name)
                      (multiple-value-bind (value exists) (gethash slot-name (act-r-chunk-slot-value-lists chunk))
                        (when (or exists
                            (car (no-output (sgp-fct (list :show-all-slots))))
                            (not
                              (extended-slot-name-p slot-name (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))   )
                            )
                          )
                          (list (list slot-name value))
                        )
                      )
                    )
                    (ct-slot-names (act-r-chunk-chunk-type chunk))
                  )
           )
          ))
          (setf aString (concatenate 'string aString
            (format nil "~@[~%parameters for chunk ~a:~%~:{   ~s  ~s~%~}~]~%"
                chunk-name
              (mapcar #'(lambda (param)
                   (list (act-r-chunk-parameter-name param)
                       (funcall (act-r-chunk-parameter-accessor param) x)
                   )
                 )
                 *chunk-parameters-list*
              )
            )
           ))
            (hookOutputInformation 0 "retrieval-set" "Chunk set" aString 1)
         )
        )
      )
     )
     )
)

(defun formatOutput-bufferChunk-details(bufferChunk)
 (if bufferChunk
   (progn
    (let* (
          (aString (string (format nil "~%[~a] buffer Chunk Details: " (act-r-chunk-name bufferChunk))))
          (theChunk (act-r-buffer-chunk bufferChunk))
```

```
          (chunk (get-chunk theChunk))
        )
      (if (not (null chunk))
        (progn
          (setf aString (concatenate 'string aString
            (format nil "~%Details for chunk ~S inside ~a buffer: ~:[ (~s) ~; ~* ~] ~% ~@[~S~%~]  ISA ~S ~% ~:{   ~s  ~s~%~}"
              theChunk
              (act-r-chunk-name bufferChunk)
              (eql theChunk (act-r-chunk-name chunk))
              (act-r-chunk-name chunk)
              (act-r-chunk-documentation chunk)
              (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))
              (mapcan #'(lambda (slot-name)
                 (multiple-value-bind (value exists) (gethash slot-name (act-r-chunk-slot-value-lists chunk))
                   (when (or exists
                       (car (no-output (sgp-fct (list :show-all-slots))))
                       (not
                         (extended-slot-name-p slot-name (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))   )
                       )
                     )
                     (list (list slot-name value))
                   )
                 )
               )
              (ct-slot-names (act-r-chunk-chunk-type chunk))
            ))))
          (setf aString (concatenate 'string aString

            (format nil "~@[~%parameters for chunk ~a inside buffer:~a~%~:{   ~s  ~s~%~}~]~%"
                theChunk
                (act-r-chunk-name bufferChunk)
                (mapcar #'(lambda (param)
                        (list (act-r-chunk-parameter-name param)
                            (funcall (act-r-chunk-parameter-accessor param) theChunk)
                        )
                      )
                      *chunk-parameters-list*
                ))))
            aString
          )
          " No internal Chunk"
      )
    )
  )
  " No internal Chunk"
 )
)

(defun formatOutput-buffer-list(buffers-list)
  (let ((aString (string "")))
   (if (listp  buffers-list)
     (progn
       (dolist (buffer buffers-list)
         (if (buffer-exists buffer)
           (progn
             (let* (
                  (bufferChunk    (buffer-instance buffer)    )
                 )
               (if bufferChunk
                 (setf aString (concatenate 'string aString (formatOutput-bufferChunk-details bufferChunk) ))
               )
             )
             )
           (setf aString (concatenate 'string aString " [" (coerce-string buffer)": does not exists] " ) )
         )
       )
     )
    (setf aString (concatenate 'string aString  " []" ) )
   )
   aString
 )
 )

(defun formatOutput-Production-Parameters(the-prod)
 (if the-prod
    (let ((prod-name (production-name the-prod)))
       (format nil  "~%    utility:    ~6,3F~%    u:         ~6,3F~%    at:        ~6,3F~%    reward:    ~6,3F~%"
         (no-output (caar (spp-fct (list prod-name :utility))))
         (no-output (caar (spp-fct (list prod-name :u))))
         (no-output (caar (spp-fct (list prod-name :at))))
         (no-output (caar (spp-fct (list prod-name :reward))))
       )
    )
 )
)

(defun formatOutput-Production-Variables(the-prod)
 (if the-prod
    (let (
         (aString (string ""))
         (variables-list (production-bindings the-prod))
       )
       (dolist (variable variables-list)
         (setf aString (concatenate 'string aString   (format nil "~%~a: ~a" (car variable)  (cdr variable))   ))
       )
       aString
    )
 )
)
```

```
(defun formatOutput-Production-Details-From-Name(prod-name)
 (has-current-mp
  "No Meta process present."
  (has-current-model
 "No model loaded"
 (let*
    (
     (the-prod (get-production (intern prod-name) ) )
     (left-hand-side (production-lhs the-prod) )
     (right-hand-side (production-rhs the-prod) )
     (lhs-buffer-list (production-lhs-buffers the-prod))
     (rhs-buffer-list (production-rhs-buffers the-prod))
    )

      (format nil " Production: ~a
         ~%Production Name: ~%~a
         ~%Production Variables: ~%~a
         ~%Production bindings:~a
         ~%Production parameters:~a
         ~%Internal chunks details for buffers [LHS] >>>>>:~% ~a
         ~%Internal chunks details for buffers [RHS] >>>>>:~% ~a
         ~%Production LHS: ~%~a
         ~%Production RHS: ~%~a
         ~%Whole Production: ~%~a
        "
      prod
      (production-name the-prod)
      (production-variables the-prod)
      (formatOutput-Production-Variables the-prod)
      (formatOutput-Production-Parameters the-prod)
      (formatOutput-buffer-list lhs-buffer-list)
      (formatOutput-buffer-list rhs-buffer-list)
      left-hand-side
      right-hand-side
      the-prod
       )
   )
  )
 )
)

(defun formatOutput-Production-Details(the-prod prod &rest headers)
 (let*
     (
      (left-hand-side (production-lhs the-prod) )
      (right-hand-side (production-rhs the-prod) )
      (lhs-buffer-list (production-lhs-buffers the-prod))
      (rhs-buffer-list (production-rhs-buffers the-prod))
     )
     (hookOutputInformation 0 (coerce-string (first headers)) (coerce-string (second headers)) (coerce
       (format nil " Production: ~a
          ~%Production Name: ~%~a
          ~%Production Variables: ~%~a
          ~%Production bindings:~a
          ~%Production parameters:~a
          ~%Internal chunks details for buffers [LHS] >>>>>:~% ~a
          ~%Internal chunks details for buffers [RHS] >>>>>:~% ~a
          ~%Production LHS: ~%~a
          ~%Production RHS: ~%~a
          ~%Whole Production: ~%~a
            "
      prod
      (production-name the-prod)
      (production-variables the-prod)
      (formatOutput-Production-Variables the-prod)
      (formatOutput-Production-Parameters the-prod)
      (formatOutput-buffer-list lhs-buffer-list)
      (formatOutput-buffer-list rhs-buffer-list)
      left-hand-side
      right-hand-side
      the-prod
     )
     'string ) (third headers))
  )
)

(defun IDTR-cycle-hook (prod)
(if (null prod)
 (hookOutputInformation 0 "Cycle-hook" "Production Firing:" "Production NULL" 1)
 (progn
 (let ( (the-prod (get-production prod) ))
    (formatOutput-Production-Details the-prod prod "cycle-hook" "Production Firing:" 1)
 )
 )
 )
)

(defun IDTR-conlict-resolution-hook (cs)
(if (null cs)
 (hookOutputInformation 0 "Conflict-resolution" "Winning Production:" "Production NULL" 1)
 (progn
 (let ( (the-prod (get-production (car cs))))
    (formatOutput-Production-Details the-prod (car cs) "Conflict-resolution" "Winning Production:" 1)
 )
 )
 )
 nil
)
```

```
(defun IDTR-retrieved-chunk-hook (rc)
 (let (
       (chunk (get-chunk rc))
       (theChunk (act-r-buffer-chunk chunk))
       (aString (string (format nil "Buffer: ~a" chunk-name)))
    )
    (progn
     (setf aString (concatenate 'string aString
      (format nil "~%~%Details for chunk ~a: ~:[ (~s) ~; ~* ~] ~% ~@[~S~%~]  ISA ~S ~% ~:{   ~s  ~s~%~}"
          rc
          (eql chunk-name (act-r-chunk-name chunk))
          (act-r-chunk-name chunk)
          (act-r-chunk-documentation chunk)
          (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))
          (mapcan #'(lambda (slot-name)
              (multiple-value-bind (value exists) (gethash slot-name (act-r-chunk-slot-value-lists chunk))
                 (when (or exists
                           (car (no-output (sgp-fct (list :show-all-slots))))
                           (not
                              (extended-slot-name-p slot-name (act-r-chunk-type-name (act-r-chunk-chunk-type chunk))   )
                           )
                       )
                       (list (list slot-name value))
                 )
              )
          )
          (ct-slot-names (act-r-chunk-chunk-type chunk))
       )
      )
     ))
     (setf aString (concatenate 'string aString
      (format nil "~@[~%parameters for chunk ~a:~%~:{   ~s  ~s~%~}~]~%"
          rc
          (mapcar #'(lambda (param)
             (list (act-r-chunk-parameter-name param)
                (funcall (act-r-chunk-parameter-accessor param) rc)
             )
          )
          *chunk-parameters-list*
       )
      )
     ))
    (hookOutputInformation 0 "retrieval-set" "Chunk set" aString 1)
    )
 )
)

(defstruct IDTR-module
  esc
  model-name
)

(defun reset-IDTR-module (IDTR-mod)
  (when IDTR-mod
      (sgp
      :conflict-set-hook      IDTR-conlict-resolution-hook
      :retrieval-request-hook  IDTR-retrieval-request-hook
      :retrieval-set-hook     IDTR-retrieval-set-hook
      :cycle-hook             IDTR-cycle-hook
      :retrieved-chunk-hook    IDTR-retrieved-chunk-hook
      :chunk-merge-hook       IDTR-chunk-merge-hook
      :chunk-add-hook         IDTR-chunk-add-hook
      )
  ))

(defun create-IDTR-module (model-name)
  (declare (ignore model-name))
  (make-IDTR-module :model-name model-name)
)

(defun delete-IDTR-module (IDTR-mod)
  (declare (ignore IDTR-mod))
)

(defun IDTR-params (IDTR-mod param)
)

(defun IDTR-module-queries (value)
)

(defun define-IDTR-module()
  (clear-all)
  (define-module-fct 'IDTR-MODULE nil

    (list (define-parameter :esc :owner nil))

    :params       'IDTR-params
    :creation     'create-IDTR-module
    :reset        '(nil reset-IDTR-module)
    :version      "2.0"
    :documentation  "A module to handle the IDTR module"
    :delete       'delete-IDTR-module
  )
)

(defun undefine-IDTR-module()
  (clear-all)
  (undefine-module-fct 'IDTR-MODULE)
```

```
)
```

Executer helper code

```
(defun cmd-executerLauncher (cmd args)
 (ignore-errors
  (cmd-executer cmd args)
 )
)

(defun coerce-string (thing)
  "Make a thing (number, symbol, string) into a string."
  (princ-to-string thing)
)

(defmacro gen-prod (goal name)
 `(gentemp (concatenate 'string (coerce-string ,goal) (coerce-string ,name)  ))
)


(defmacro genp (goal name)
 `(gentemp (concatenate 'string (coerce-string ,goal) "-" (coerce-string ,name) "-"))
)


 (defmacro gent (goal name)
 `(gentemp (concatenate 'string (coerce-string ,goal) "-type-" (coerce-string ,name) "-")))

(defun time-string ()
 (multiple-value-bind (s m h)  (get-decoded-time)
        ;(DECODE-UNIVERSAL-TIME (GET-UNIVERSAL-TIME)) ;(get-decoded-time);(get-universal-time);
  (format nil "~A:~2,,,'0@A:~2,,,'0@A" (+ 1 h) m s))
)

(defun string-split (split-string astring)
 "Returns a list containing items in 'string' split from occurrences of 'split-string'."
 (loop with l = (length split-string)
  for n = 0 then (+ pos l)
  for pos = (search split-string astring :start2 n)
   if pos
    collect (subseq astring n pos)
   else
    collect (subseq astring n)
  while pos
 )
)

(defun string-to-list (s)
 (assert (stringp s) (s) "~s should be a string but is not")
 (coerce s 'list)
)

(defun convert-strings-integer(my-list)
 (loop for x in my-list
  collect (locale-parse-number x)
 )
)

(defun vector-to-list (vec)
 (let* (
    (new-list nil)
  )
  (dotimes (i (length vec))
   (push (aref vec i) new-list)
  )
  (reverse new-list)
 )
)


(defun list-to-vector-a (list)
 (let* (
  (length (length list))
   (vec (make-array length))
  )
  (dotimes (i length)
   (setf (aref vec i) (pop list))
  )
  vec
 )
)

(defun string-arguments-to_list (args)
 (let* (
  (new-list (string-split " " args))
  (new-list-length (length new-list))
  (my-vector (list-to-vector-a new-list))
  (my-vector-length (length my-vector))
  (new-object-vector (make-array new-list-length) )
  )
  (dotimes (i my-vector-length)
   (let* (
     (vector-item (aref my-vector i))
     (item-bits (string-split ":" vector-item))
    )
    (cond
     ((string= "s" (first item-bits))
      (setf (aref new-object-vector i) (second item-bits))
```

```
        )
        ((string= "i" (first item-bits))
         (setf (aref new-object-vector i) (parse-integer (second item-bits)))
        )
       )
      )
    )
    (vector-to-list  new-object-vector)
  )
)

(defun cmd-executer (operation &rest argument-list)
   (let* (
         (timing)
         (args (string-arguments-to_list (first argument-list)))
         (args-length (length args))
         (newOp (symbol-function  (find-symbol (string-upcase operation))))
        )
        (if (equal (first args) NIL)
         (progn
          (setq timing (apply newOp nil ) )
         )
         (progn
          (setq timing (apply newOp args )  )
         )
        )
     (format nil "~a" timing)
   )
)

 (defun cmds-to_list (args)
    (let* (
          (timing)
          (new-list (string-split "@" args))
          (new-list-length (length new-list))
          (my-vector (list-to-vector-a new-list))
          (my-vector-length (length my-vector))
          (new-object-vector (make-array new-list-length) )
         )
       (setq timing 0.0)
       (dolist (item new-list)
         (let  (
                (newcmd-list (string-split "^" (coerce-string item) ))
               )
               (declare (dynamic-extent newcmd-list ))
               (setq timing (+ timing (cmd-executer (car newcmd-list) (car (cdr newcmd-list)))) ) )
         )
         timing
    )
)

(defun thenewtestfunction (data)
 (if (not(equal (length data) 0))
     (cmds-to_list data)
 )
)
```

# APPENDIX F: MUTATION SYSTEM PUMP

```
int GANewMutator::IDTR_Mutator(GAGenome & c, float pmut){
	genesPtr = &(GA1DArrayAlleleGenome<float>&)c; //retreive the genes from Galib
	//retreive the user data were some metrics and pointers are kept
	data = (UserData *)genesPtr->userData();
	genesSize = genesPtr->size();//get number of genes
	//Create some masks to hold some status about genes and mutator selection
//the muationsMask is passed to retrieve from the operators
//which actual genes was affected by the mutation system
	mutationsMask = (int*) calloc (genesSize,sizeof(int));
	mutationMaskFilter = (int*) calloc (genesSize,sizeof(int));
	//if we do have created thos masks
	if (mutationsMask!=NULL && mutationMaskFilter != NULL){
		//Randomise the gene status. 1 means they will undergo mutation. 0 otherwise
		genesAffected = 0;
	  if(GAFlipCoin(0.95)){ //a small chance that only one gene will be mutated. i.e. 5%
			for (i = 0; i < genesSize; i+=2){ //if a selection of selection of genes is selected
				//flip a coin to decide the status taking the mutation rate into account
				if(GAFlipCoin(pmut )){
					//a pair of adjacent genes are going to be mutated i.e. X&Y coordinated
					mutationMaskFilter[i] = 1;mutationMaskFilter[i+1] = 1;
				}else{
					mutationMaskFilter[i] = 0;mutationMaskFilter[i+1] = 0;
				}
//increasse the amount index if selected genes for mutation
					genesAffected += 2;
			}
		}else{
			//if one a single gene is selected, get a random  X coordinate from the pool
			int mutatatedGeneIndex = IDTR_GAEvenRandomInt(0, genesSize);
  for (i = 0; i < genesSize; i+=2){ //make that all are turned off
					mutationMaskFilter[i] = 0;mutationMaskFilter[i+1] = 0;
			}
//and turn the selected pair on
			mutationMaskFilter[mutatatedGeneIndex] = 1;mutationMaskFilter[mutatatedGeneIndex+1] = 1;
		}
		//randomise which mutator to fire using masks from our pool of 12 operators
		int nbMutators = 12;
		//get a mutation operator at random
		//int mutatorIndex = mutatorIndex = GARandomInt(0,nbMutators-1);
    int mutatorIndex = 0;
		if(!data->configInstance->goodLayoutFound) //if a workable layout not found
			mutatorIndex = GARandomInt(0,nbMutators-1); //then get any operator at random
		else{
			if (GAFlipCoin(0.50)){ //if the toss of a coing is favorable, select any oprator
				mutatorIndex = GARandomInt(0,nbMutators-1);
			} else{//otherwise
				if(GAFlipCoin(0.50)) //toss a coin to select which of the sticking operator to select
					mutatorIndex = 10; //stick to a related component by an action
				else
					mutatorIndex = 9; //stick to any component in the layout
			}
		}
		geneMutationProb = pmut; //ensure that the operators are getting the mutationrate
		//randomise the genes that will be mutated
		randomizer.setup(&geneMutationProb, &genesSize, mutationMaskFilter);
//retreive from the randomiser a pointer to the genes to pass to the operators
		genesLoc = randomizer.getValuesVecPtr();
//ensure that the oprators know how mny genes will be affected
		genesLocSize = genesLoc->size();
mutationOperators[mutatorIndex](); //invoke the selected mutator
		//retreive the total number of mutation to return to Galib for tis stats
		mutations = 0;
		for (int i = 0; i < genesSize; i++){
			if (mutationsMask[i] > 0)
				mutations += mutationsMask[i];
		}
		//clean up
		if (mutationMaskFilter) free (mutationMaskFilter);
		if (mutationsMask) free (mutationsMask);
//and return the number of mutations that took place to Galib
		return mutations;
	}
	return 0;
}
```

# APPENDIX G: BLOCK RANDOM MUTATION OPERATOR

```
void RandomBlockComponentsShift(){
	//get random horizontal, vertical direction
	leftORright = (GALEFTRIGHT) GARandomBit(); upORdown = (GAUPDOWN)GARandomBit();
	direction = (GADIRECTION)GARandomInt(0,2);
	w = data->width; h = data->height; //retreive metrics of the layout
	//compute random nudging value
	offsetXVal = _GARandomFloat_(&minFloatRandom,&w); offsetYVal = _GARandomFloat_(&minFloatRandom,&h);
	while (genesLocSize > 0){ // while there are still a number of genes pairs to mutate
		//retreive both X and Y index for coordinate of a component
		destX = randomizer.getRandomNoReplacement();
		genesLocSize = randomizer.getNumberValuesInVector();
		destY = destX + 1;
		if (destX < 0) break; //if errors break away to terminate process
		//reteive the actual value of component coordinates i.e. gene values
		valX = newValX = genesPtr->gene(destX); valY = newValY = genesPtr->gene(destY);
		//and retrieve the metrics of the component i.e. width and height
		XcompMetric = data->GUI_metrics[destX]; YcompMetric = data->GUI_metrics[destY];
		//if direction is horizontal i.e. going left or right
		if(direction == GOING_HONRIZONTAL){
			//if left was selected
			if (leftORright == GOING_LEFT){
				newValX = valX - offsetXVal; //shift component to left
				//deal with situation when outside the layout
				//by shifting by a reflective amount the other side
				if (newValX < 0) newValX = abs(newValX) + 1;
			}else{//if going right
				newValX = valX + offsetXVal; //shift component to right
				//deal with situation when outside the layout
				if (newValX + XcompMetric > w) newValX = newValX - ((newValX + XcompMetric) - w) - 1;
			}
			genesPtr->gene(destX, newValX); //save new location in the chromosome
			mutationsMask[destX]++; //and increase mutation number
		//if the direction is to go vertically
		}else if (direction == GOING_VERTICAL){
			if (upORdown == GOING_UP){ //if selected direction is to go upward
				newValY = valY - offsetYVal; //shift component upward
				if (newValY < 0) newValY = abs(newValY) + 1; //and deal with situation when outside the layout
			}else{//if doing downward
				newValY = valY + offsetYVal; //shift component downward
				//and deal with situation when outside the layout
				if (newValY + YcompMetric > h) newValY = newValY - ((newValY + YcompMetric) - h) - 1;
			}
			genesPtr->gene(destY, newValY); //save new location in the chromosome
			mutationsMask[destY]++; //and increase mutation number
}			else if (direction == GOING_DIAGONAL){ //if the direction is to go diagonally
			if (leftORright == GOING_LEFT){ //if one of selected direction is to left
				newValX = valX - offsetXVal; //shift component on the left
				if (newValX < 0) newValX = abs(newValX) + 1; //and deal with situation when outside the layout
			}else{{//if doing on the right
				newValX = valX + offsetXVal; shift component to the right
				//and deal with situation when outside the layout
				if (newValX + XcompMetric > w)
				   newValX = newValX - ((newValX + XcompMetric) - w) - 1; //newValX - (w-newValX) - 1;
			}
			if (upORdown == GOING_UP){ {{//if doing on upward
			   newValY = valY - offsetYVal; shift component upward
				//and deal with situation when outside the layout
				if (newValY < 0) newValY = abs(newValY) + 1;
			}else{ {{//if doing downward
				newValY = valY + offsetYVal; //shift component downward
				//and deal with situation when outside the layout
				 if (newValY + YcompMetric > h)
				   newValY = newValY - ((newValY + YcompMetric) - h) - 1; //newValY - (newValY - h) -1;
			}
			//save new location in the chromosome
			genesPtr->gene(destX, newValX);
			genesPtr->gene(destY, newValY);
			//and increase mutation number
			mutationsMask[destX]++;
			mutationsMask[destY]++;
			}
	}
}
```

# APPENDIX H: TOISE 2-POINTS CROSSOVER

```
int GACrossovers::IDTRTwoPointPairCrossover(const GAGenome& p1, const GAGenome& p2, GAGenome* c1, GAGenome* c2){
	//retrieve the genome from Galib
	const GA1DArrayGenome<float>& mom=DYN_CAST(const GA1DArrayGenome<float>&, p1);
	const GA1DArrayGenome<float>& dad=DYN_CAST(const GA1DArrayGenome<float>&, p2);
	int ofsprings = 0; float geneX = 0.0f; float geneY = 0.0f;
	//retrieve Mum and Dad chomosome length
	mumLength = mom.length();
	dadLength = dad.length();
	//get a random value for cutting location
	pos1 = GAUtils::getEvenRandomINT(&zeroIntVal, &mumLength);
	pos2 = GAUtils::getEvenRandomINT(&zeroIntVal, &mumLength);
	//if genes properly retreived and we have two of them i.e. need 2 ofsprings
	//NOTE: be careful that some type of GA only generate 1 ofspring
	if(c1 && c2){
		//cast the genomes to buffers of genes
		GA1DArrayGenome<float> &sis=DYN_CAST(GA1DArrayGenome<float>&, *c1);
		GA1DArrayGenome<float> &bro=DYN_CAST(GA1DArrayGenome<float>&, *c2);
		//if it happens that random values are not the right way, swap them
		if (pos1 > pos2)
			SWAP(pos1,pos2);
		//compute length of segment
		int len  = pos2-pos1;
		//toss a coin to decide which way round the infomration is transfer to next generation
		if(GARandomBit()){
			//copy gene information from parents to one child i.e. MUM, DAD,MUM
			sis.copy(mom, 0, 0, pos1);
			sis.copy(dad, pos1, pos1, len);
			sis.copy(mom, pos2, pos2, mumLength-pos2);
			//repeat process for other child i.e. DAD, MUM,DAD
			bro.copy(dad, 0, 0, pos1);
			bro.copy(mom,pos1, pos1, len);
			bro.copy(dad, pos2, pos2, dadLength-pos2);
		}else{
			//copy gene information from parents to one child i.e. DAD, MUM,DAD
			sis.copy(dad, 0, 0, pos1);
			sis.copy(mom, pos1, pos1, len);
			sis.copy(dad, pos2, pos2, dadLength-pos2);
			//repeat process for other child i.e. MUM, DAD,MUM
			bro.copy(mom, 0, 0, pos1);
			bro.copy(dad,pos1, pos1, len);
			bro.copy(mom, pos2, pos2, mumLength-pos2);
		}
		//tell Galib 2 ofsprings were generated
		ofsprings = 2;

	}
	//if only 1 ofspring is needed
	else if(c1 || c2){
		//cast the genomes to buffers of genes
	GA1DArrayGenome<float> &kid = (c1 ? DYN_CAST(GA1DArrayGenome<float> &, *c1) :
					DYN_CAST(GA1DArrayGenome<float> &, *c2));
		//if it happens that random values are not the right way, swap them
		if (pos1 > pos2)
			SWAP(pos1,pos2);
		//compute length of segment
		int len  = pos2-pos1;
		toss a coin to decide which way round the infomration is transfer to next generation
		if(GARandomBit()){
			//copy gene information from parents to one child i.e. MUM, DAD,MUM
			kid.copy(mom, 0, 0, pos1);
			kid.copy(dad, pos1, pos1, len);
			kid.copy(mom, pos2, pos2, mumLength-pos2);
		}
		else{
			copy gene information from parents to one child i.e. DAD, MUM,DAD
			kid.copy(dad, 0, 0, pos1);
			kid.copy(mom, pos1, pos1, len);
			kid.copy(dad, pos2, pos2, mumLength-pos2);
		}
		//tell Galib 1 ofsprings were generated
		ofsprings = 1;
	}
	return ofsprings;
}
```

# APPENDIX I: GA PENALTY ALGORITHMS

```
float JavaObjectsManager::getAreaOverlapPenaltiesFactor(){

 long interceptAreaCumul = 0.0f;
 JavaComponent* compA = NULL;
 register int i;
 register int j;
 long totalArea = 0;
 std::vector<JavacompSymbols> symbolsVec;

 for (compA = (JavaComponent*)firstComponent; compA; ){
  /*
   We need to ensure that static components are not taken into account and
        that they are allowed to move on the layout
   Any other components are saved to a symbol that will calculate anything needed et
  */
  if (compA && compA->allowed
     && (isComponentAllowedToMove(&compA->baseClass, &compA->className)
        || compA->className.compare("javax.swing.JMenuBar") == 0)

  ){
   JavacompSymbols symbol;
   //get the component metrics into the symbol
   compA->fillSymbol(symbol);
   //increase ID
   compTotal ++;
   symbol.id = compTotal;
   //push it into a vector
   symbolsVec.push_back(symbol);
   }
   //next component
   compA = (JavaComponent*)compA->nextComponent;
  }

  //we now detect the amount of interceptions for each components
  for (i = 0; i < (int)symbolsVec.size(); i++){
   for (j = 0; j < (int)symbolsVec.size(); j++){

    //ensure that we do not into account a component overlapping with itself
    if (symbolsVec[j].id == symbolsVec[i].id)
     break;

    //increase the total area by the area of this component
    totalArea += symbolsVec[j].area;

    //model the GUI components as a CRECT objects so we can use its functionality
    rectA = CRect(symbolsVec[i].x, symbolsVec[i].y,
                  symbolsVec[i].x + symbolsVec[i].width, symbolsVec[i].y + symbolsVec[i].height);
    rectB = CRect(symbolsVec[j].x, symbolsVec[j].y,
                  symbolsVec[j].x + symbolsVec[j].width, symbolsVec[j].y + symbolsVec[j].height);

    //get the intersection CRECT for both object to compare for overlap
    interRes = intersecRect.IntersectRect(rectA, rectB) == 1;

    //if overlap it exists
    if (interRes){

     //calculate the intersection area
     interceptArea = intersecRect.Width() * intersecRect.Height();

     //and increase the total overlap area
     interceptAreaCumul += interceptArea;
    }
   }
  }
  //and return the fraction of overlap compared to total component area
  return  (interceptAreaCumul * 1.0f)/(totalArea * 1.0f);
}



float JavaObjectsManager::getOutofAreaPenalties(){

//do a little checks to beging with
        configInstance->javaManagerBusy = true;
        if(areasManager->getNumberAreas() == 0)
                return 0.0f;
        penalties = 0;
        //for each component in the layout
        for (comp = (JavaComponent*)firstComponent; comp; ){
                crectA.SetRect(comp->x, comp->y,comp->x+comp->width, comp->y + comp->height);
//if the component is bound to an area
                if(comp->bindedToArea){
//get the area
                        area = areasManager->getAreaFromID(&comp->areaID);
                        //if we have one
                        if (area){
//translate this area to a CRect component and apply the penalties
                                areaRect.SetRect(area->topLeft,area->bottomRight);
                                if (crectA.left < areaRect.left || crectA.right > areaRect.right
                ||crectA.top < areaRect.top ||crectA.bottom > areaRect.bottom){
                                        penalties += 6;
                                if(areaRect.PtInRect(crectA.CenterPoint()))
                                        penalties -= 1;
                                if(areaRect.PtInRect(area->bottomRight))
                                        penalties -= 1;
                                if(areaRect.PtInRect(CPoint(area->topLeft.x, area->bottomRight.y)))
```

```
                                        penalties -= 1;
                                if(areaRect.PtInRect(CPoint(area->bottomRight.x, area->topLeft.y)))
                                        penalties -= 1;
                                }
                        }

                        //the component make be in an area it is not supposed to be in
//so attempt to get it
                        Area* OtherArea = areasManager->getAreaForPoint(CPoint(crectA.CenterPoint()));
                        CRect intersecRect;
                        //if it exists
                        if(OtherArea){
                                //and we have an area
                                if(area){
                                        //make sure we are not penalising for being in tis own area! ☺
                                        if(OtherArea->getID() != area->getID()){
                                                //if if there is any intersect
                                                CRect crectB;
                                                CPoint p1 = OtherArea->topLeft;
                                                CPoint p2 = OtherArea->bottomRight;
                                                crectB.SetRect( OtherArea->topLeft, OtherArea->bottomRight);
                                                //if included increase the peanlties points
                                                bool interRes = intersecRect.IntersectRect(areaRect, crectB) == 1;
//and if there is penalise this component
                                                if(interRes)
                                                   penalties += 6;
                                        }
                                }
                        }
                }
//next component
                comp = (JavaComponent*)comp->nextComponent;
        }
        //we are not doing anything anymore
        configInstance->javaManagerBusy = false;
        //return overall penalties
        return penalties ;
}
```

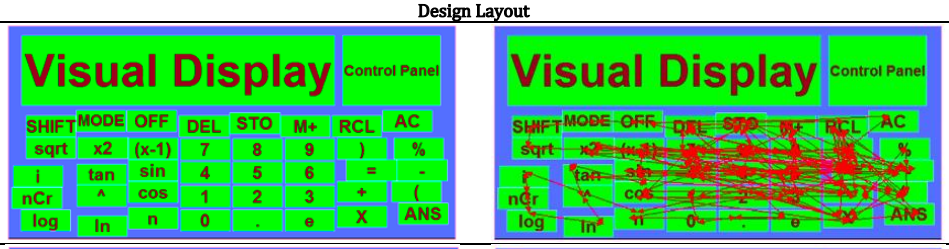
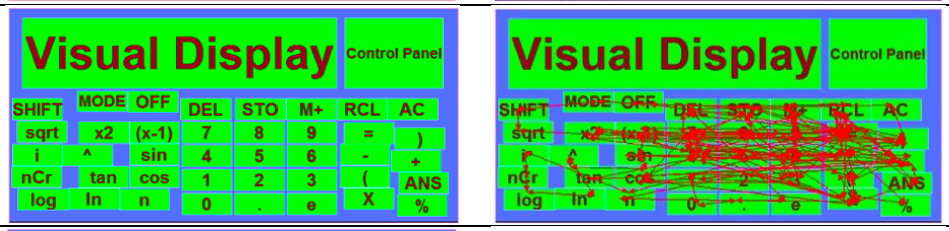
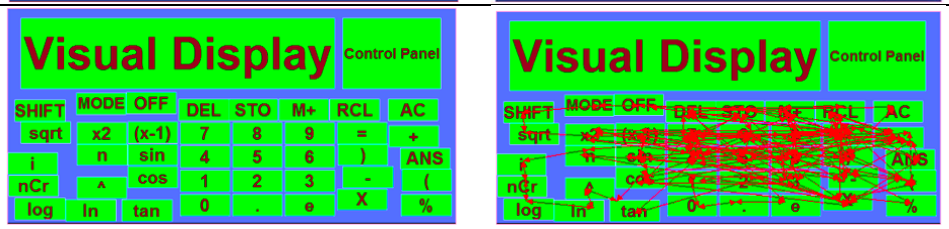
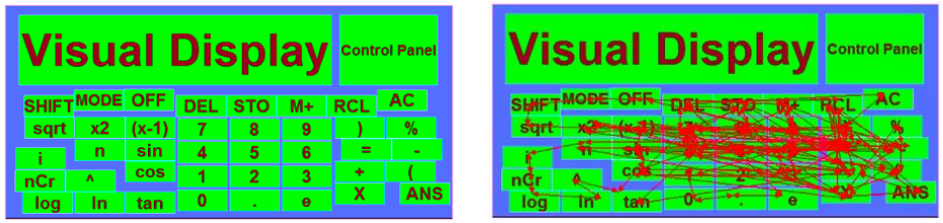
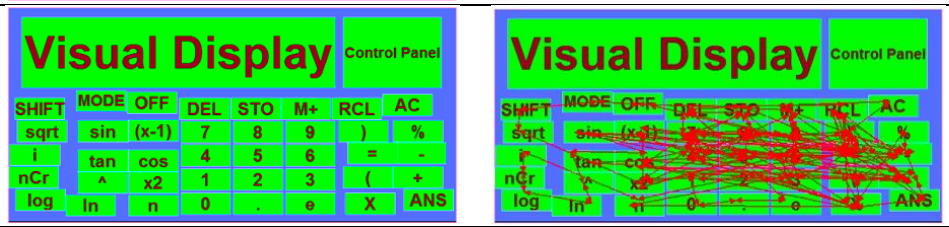
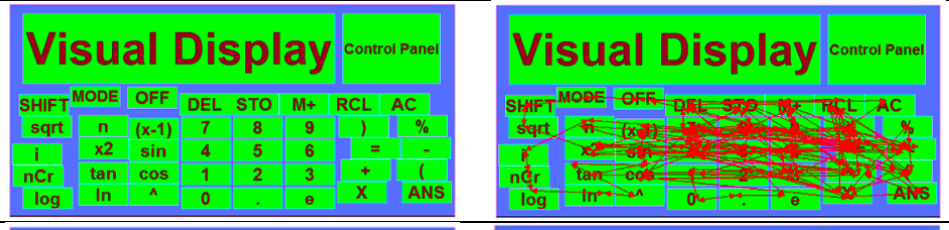
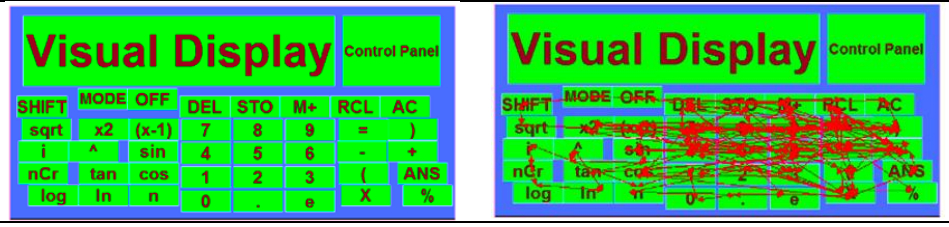
# APPENDIX J: GENETIC ALGORITHM SURROGATE METHOD EXAMPLE

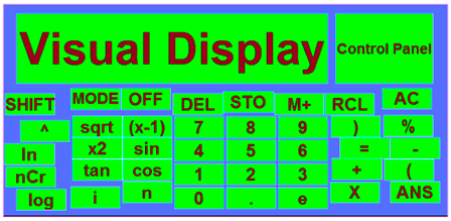
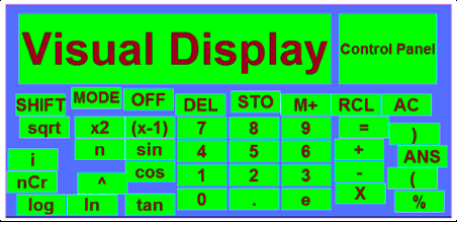
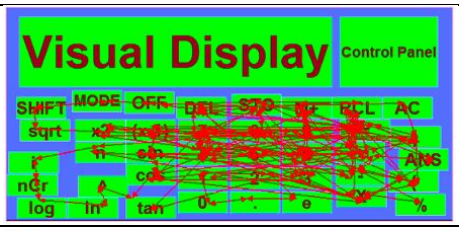
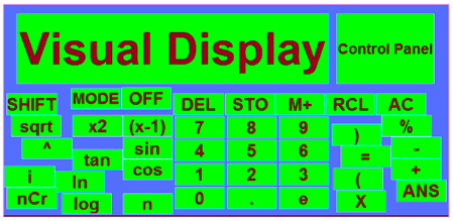
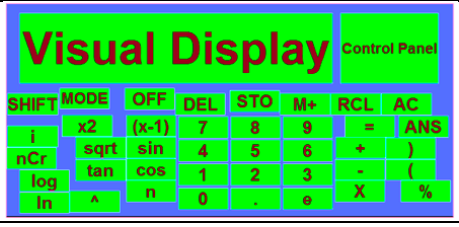
```
inline float timingDragDropActionMoveTo(ActionRecord* actionRecord){

timing = 0.0f; cartesianLength = 0.0f; angleForCartesianDistance = 0.0f;
approachWidth = 0.0f; lengthDegrees = 0.0f; newFitts_coefficient = 0.0f;
last_x =          actionRecord->action1_x; last_y = actionRecord->action1_y;
lastWidth =       actionRecord->action1_width;lastHeight = actionRecord->action1_height;
from_x =          actionRecord->action2_x; from_y = actionRecord->action2_y;
fromWidth =       actionRecord->action2_width; fromHeight = actionRecord->action2_height;
to_x =            actionRecord->action3_x; to_y = actionRecord->action3_y;
toWidth =         actionRecord->action3_width; toHeight = actionRecord->action3_height;
fitts_time = 0.0f; prep_time = 0.05f;
lastAngleForCartesianDistance = 0.0f; lastLengthDegrees = 0.0f;
start_time = 0.485f; dir = false;
dist = false; finish_time = 0.0f; //(+ (burst-time module) (exec-time mvmt))
exec_time = 0.0f;                 //(+ (init-time mtr-mod) fitts )
realFittsVal = 0.0f; nfeats = 0;

//the first action will be a cursor one
//cursor-ply is based on ply itself descendant from hfrt-movement which style is:HAND_FINGER_R_THETA
actionStyle = HAND_FINGER_R_THETA; lastActionStyle = NOSTYLE;
styleFeatures = featureNumberForActionStyle[actionStyle];

//FOCUSING AND MOVING OVER THE FIRST ORIGN TARGET I.E. last_x,last_y,last_x-1,last_y-1
//WE MOVE TO THE TARGET A AND WE HAVE SET UP THE CURSOR AND HAND POSITION TO TARGET A MINUS 1 FOR
BOTH X //AND Y METRICS
//WE DO THIS QUICK CALL TO GET THE ANGLE AND LENGTH SO IT CAN BE USED TO CALCULATE THE DIFFERENCE IN
//FEATURES FOR THE NEXT ACTIONS
last_x1 = last_x-1; last_y1 = last_y-1;
move_cursor_calculations(cartesianLength,angleForCartesianDistance,
        approachWidth,lengthDegrees,&lastWidth,&lastHeight,&last_x1,&last_y1,&last_x,&last_y);
newFitts_coefficient = fitts_coefficient_func();
fitts_time = fittsFunc(realFittsVal, &lengthDegrees, &approachWidth, &newFitts_coefficient);
        nfeats = getNumberDifferentFeatures(dir,dist,&lastActionStyle,&actionStyle,
        &lastAngleForCartesianDistance,&angleForCartesianDistance,&lastLengthDegrees,&lengthDegrees)
        ;
lastAngleForCartesianDistance = angleForCartesianDistance; lastLengthDegrees = lengthDegrees;
//THE STYLE REMAINS THE SAME
lastActionStyle = actionStyle; actionStyle = HAND_FINGER_R_THETA;
//WE ARE MOVING TOWARD TARGET A. THIS SHOULD BE LIKE THIS NOT PASSING THE HEIGHT INSTEAD OF THE WIDTH
move_cursor_calculations(cartesianLength,angleForCartesianDistance,approachWidth,lengthDegrees,&fromW
idt
        h,&fromHeight,&last_x,&last_y,&from_x,&from_y);
newFitts_coefficient = fitts_coefficient_func();
fitts_time = fittsFunc(realFittsVal, &lengthDegrees, &approachWidth, &newFitts_coefficient);
nfeats = getNumberDifferentFeatures(dir,dist,&lastActionStyle,&actionStyle,
        &lastAngleForCartesianDistance,&angleForCartesianDistance,&lastLengthDegrees,&lengthDegrees)
;
prep_time = computePrepTime(nfeats);
timing = 1.02f;
if (cartesianLength == 0.0f){
        finish_time = 0.285f;
        timing += finish_time;
        timing += 0.185f;
}else{
        finish_time =  fitts_time + 0.05f + 0.05f;
        timing += prep_time + finish_time;
        timing += 0.235f;
        if (prep_time == 0.0f){
                timing += 0.035f;
        }
}

timing += 0.0000001729f; timing += 0.165f; lastTiming = timing;
lastAngleForCartesianDistance = angleForCartesianDistance; lastLengthDegrees = lengthDegrees;
//THE STYLE REMAINS THE SAME
lastActionStyle = HAND_FINGER_R_THETA; actionStyle = HAND_FINGER_R_THETA;
move_cursor_calculations(cartesianLength,angleForCartesianDistance,approachWidth,lengthDegrees,&fromW
idt     h,&fromHeight,&last_x,&last_y,&from_x,&from_y);
newFitts_coefficient = fitts_coefficient_func();
fitts_time = fittsFunc(realFittsVal, &lengthDegrees, &approachWidth, &newFitts_coefficient);
nfeats = getNumberDifferentFeatures(
        dir,dist,&lastActionStyle,&actionStyle,&lastAngleForCartesianDistance,&angleForCartesianDist
ance
        ,&lastLengthDegrees,&lengthDegrees);
lastAngleForCartesianDistance = angleForCartesianDistance; lastLengthDegrees = lengthDegrees;
//THE STYLE REMAINS THE SAME
lastActionStyle = actionStyle; actionStyle = HAND_FINGER_R_THETA;
move_cursor_calculations(cartesianLength,angleForCartesianDistance,approachWidth,lengthDegrees,&fromW
idth,&fromHeight,&from_x,&from_y,&from_x,&from_y);
newFitts_coefficient = fitts_coefficient_func();
fitts_time = fittsFunc(realFittsVal, &lengthDegrees, &approachWidth, &newFitts_coefficient);
```

```
nfeats = getNumberDifferentFeatures(
dir,dist,&lastActionStyle,&actionStyle,&lastAngleForCartesianDistance,&angleForCartesianDistance,&las
tLengthDegrees,&lengthDegrees);
lastAngleForCartesianDistance = angleForCartesianDistance; lastLengthDegrees = lengthDegrees;

//WE JUST USE THE PUNCH STYLE FOR CLICKING
lastActionStyle = PUNCH; actionStyle = HAND_FINGER_R_THETA;
move_cursor_calculations(cartesianLength,lastAngleForCartesianDistance,approachWidth,lastLengthDegree
s,&
        toWidth,&toHeight,&from_x,&from_y,&to_x,&to_y);
newFitts_coefficient = fitts_coefficient_func();
fitts_time = fittsFunc(realFittsVal, &lastLengthDegrees, &approachWidth, &newFitts_coefficient);
nfeats = getNumberDifferentFeatures(dir,dist,&lastActionStyle,&actionStyle,
        &angleForCartesianDistance,&lastAngleForCartesianDistance,&lengthDegrees,&lastLengthDegrees)
;
prep_time = computePrepTime(nfeats);
if (cartesianLength == 0.0f){
        finish_time = 0.285f; timing += finish_time; timing += 0.150f;
}else{
        finish_time =  fitts_time + 0.05f + 0.05f;
        timing += prep_time + finish_time; timing += 0.050f; timing += 0.300f;
        if (prep_time == 0.0f){ timing += 0.035f;}
}
return timing;
}


inline void move_cursor_calculations(float& cartesianDistance,float& angle,float& approachWidth,
        float& angleForDistance, float* width, float* height, float* from_x, float* from_y,
        float* to_x, float* to_y){
//we first use the cartesian coodinates x1 y1 x2 y2 of our 2 points to get the distance from the
//cartesian distance between those points and the angle from origin to this tangent
xy_to_polar(angle, cartesianDistance, from_x, from_y, to_x, to_y);
angleForDistance = distanceToangle(&cartesianDistance);
//now the width in pixels in the GUI must be re-calculated with eye position and distance from screen
approachWidth = calcApproach_width(width,height,angle);

if (approachWidth <= 0.0f){
        approachWidth = 1.0f;
}
}

//this is computed by the CURSOR-PLY  in the compute-exec-time function of motor.lisp
//which seems to do that only. The cursor-ply object is created in the move-cursor function
//as part of the "prepare-movement" process
inline float fitts_coefficient_func(){
return exp(control_order) * fitts_coefficient;
}

inline float fittsFunc(float& realFittsVal, float* lengthDegrees, float* approachWidth,
        float* fittsCoefficient){
realFittsVal = *fittsCoefficient *(log((*lengthDegrees/*approachWidth)+0.5)/log(2.0f));
result = realFittsVal;
if (result < min_fitts_time)
        result =min_fitts_time;
return result;
}

inline float calcAngle(float* p1_x, float* p1_y, float* p2_x, float* p2_y){
a_x = *p2_x - *p1_x; a_y = *p2_y - *p1_y; b_x = 1.0; b_y = 0.0;
return acos((a_x*b_x+a_y*b_y)/sqrt(a_x*a_x+a_y*a_y));
}

inline void xy_to_polar (float& angle, float& distance,  float* from_x, float* from_y, float* to_x,
        float* to_y){
//we first calculate the distance between those 2 points
distance = cartesianDistance(from_x, from_y, to_x, to_y);
//and then returns the angle of to the tangent between those 2 points
if (distance == 0.0f) angle = 0.0f; else angle = atan2( *to_y-*from_y,*to_x-*from_x);
}
inline float degreesToRadian(float* degrees){
//return degrees * PI / 180.0f;
return D_toR_(*degrees);
}


inline float radianTodegrees(float* radians){
//return radians * 180 / PI;
return R_toD_(*radians);
}

inline float angleTopixels_mth(float* angle ){
float result = distanceFromScreen * tan(degreesToRadian(angle)) * screenDPI;
return result;
}
```

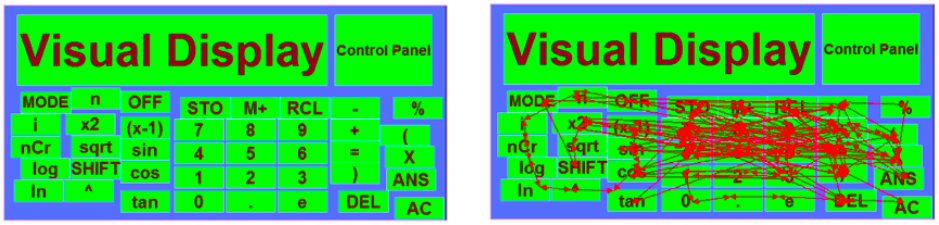
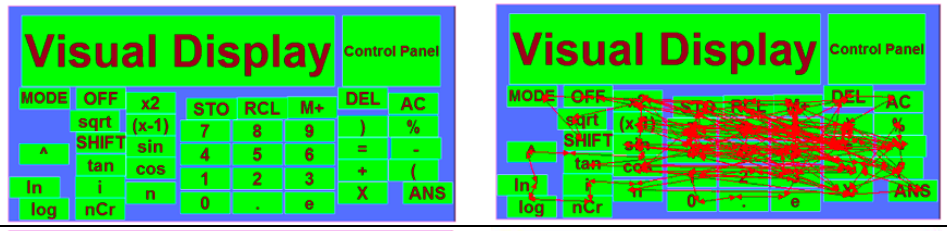
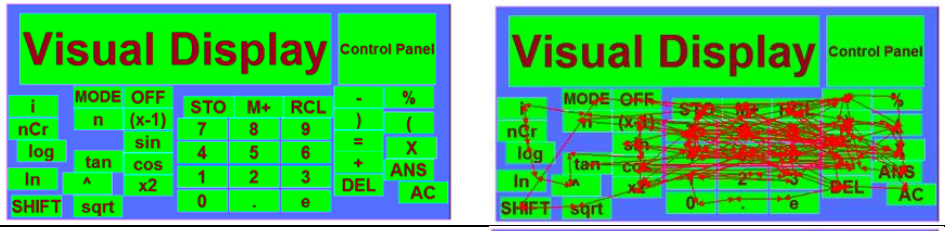
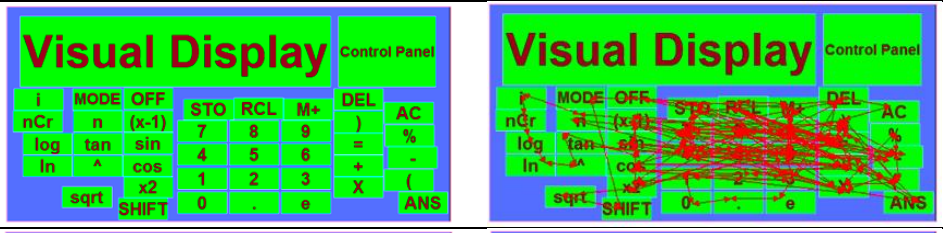
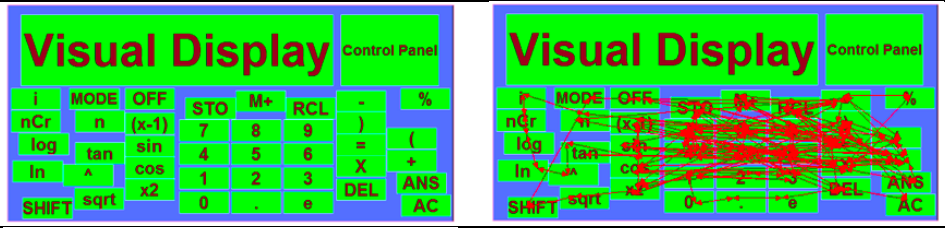
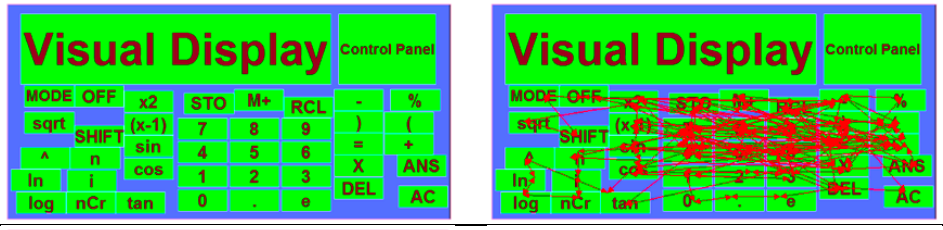
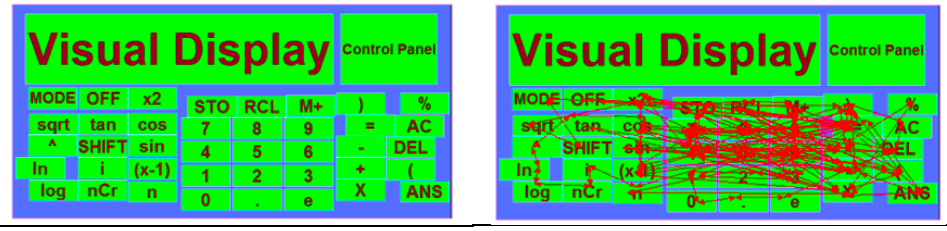
```
inline float distanceToangle(float* pixels){
//we have a distance in pixels and we are looking from a certain distance from the screen.
//We first adjust it for the screen DPI
//we can get the angle of sight using the R=atan(X/Y) where X=pixels and Y distance = //distanceFromScreen. We then return the degrees rather than radians.
distanceAngle = atan( (*pixels / screenDPI) / distanceFromScreen);
return radianTodegrees(&distanceAngle);
}

inline float cartesianDistance(float* from_x, float* from_y, float* to_x, float* to_y){
return sqrt((*to_x-*from_x)*(*to_x-*from_x) + (*to_y-*from_y)*(*to_y-*from_y));
}

inline float calcApproach_width(float* width, float* height,  float& polar_angle){
critical_theta = atan2(*height,*width); theta = abs(polar_angle);
if (theta > _PI_/2.0f) theta = _PI_ - theta;
float result = 0.0f;
if(theta == 0.0f)   result = *width;
else if(theta == _PI_/2.0f) result =  *height;
else if (theta == critical_theta)         result =  cartesianDistance(height,width,height,width);
else if (theta < critical_theta)          result =  *width / cos(theta);
else    result =  *height / cos( _PI_/2.0f - theta);
return = distanceToangle(&result);
}
```

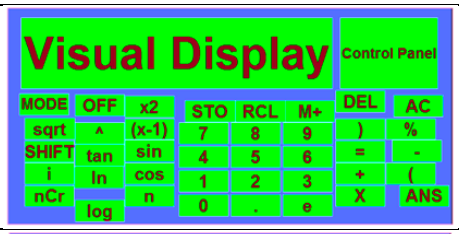
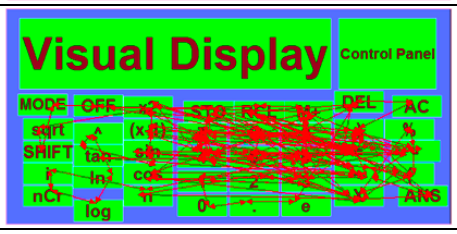
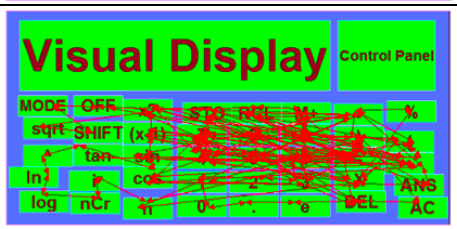
# APPENDIX K: TOP 5% 4 AREAS OPTIMISED DESIGN LAYOUTS WITH 10000 GENERATION SIZE

| Design Layout | ACT-R latency |
|---|---|
|  | 204.481 |
|  | 204.484 |
|  | 204.535 |
|  | 204.536 |
|  | 204.544 |
|  | 204.549 |
|  | 204.552 |

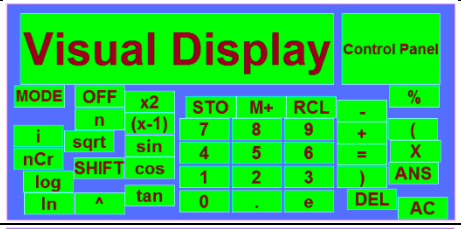
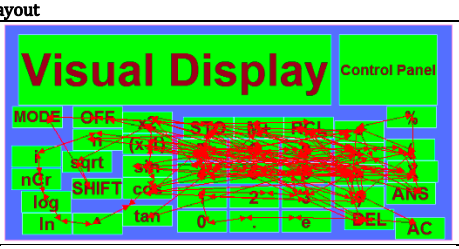
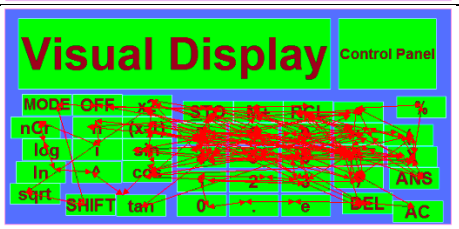
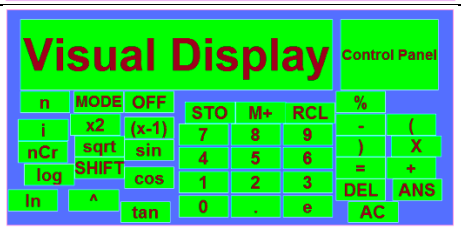
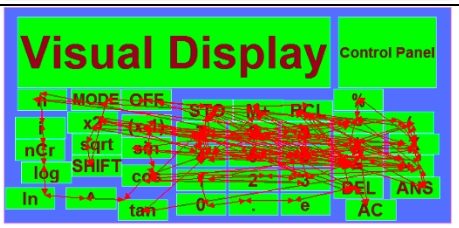
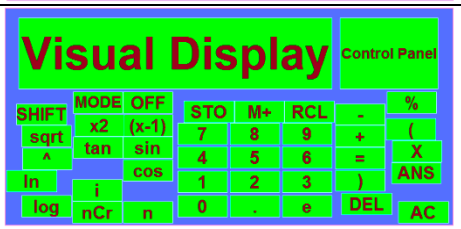
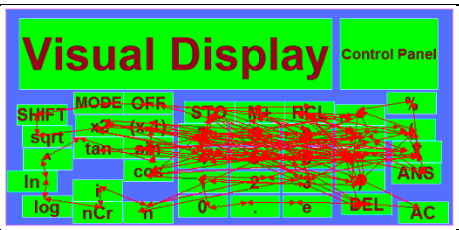
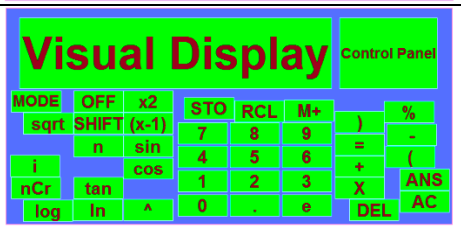
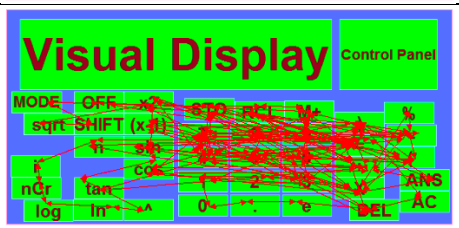
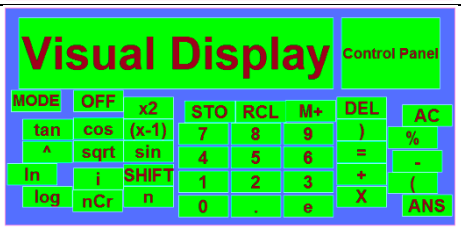
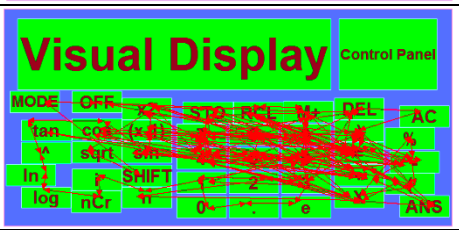
| | | |
|---|---|---|
|  |  | 204.565 |
|  |  | 204.567 |
|  |  | 204.567 |
|  |  | 204.569 |
|  |  | 204.575 |

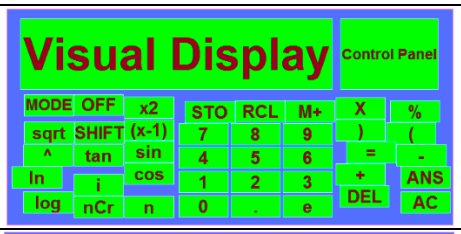
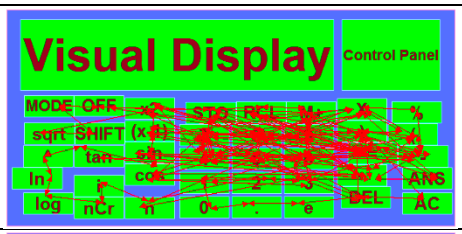
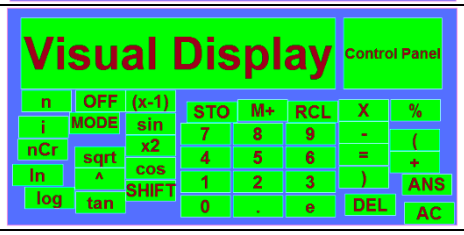
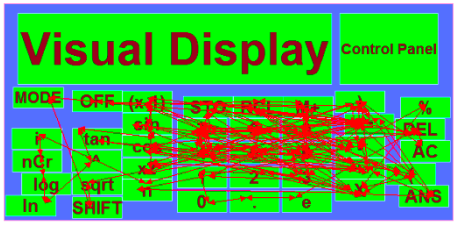
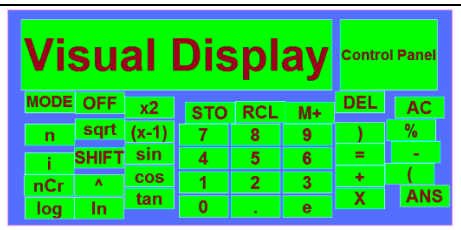
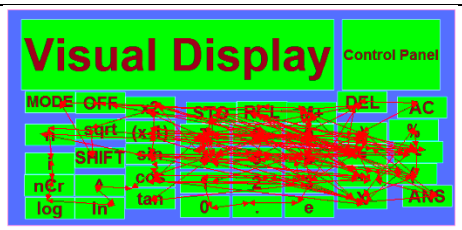
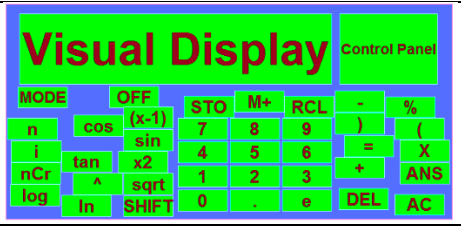
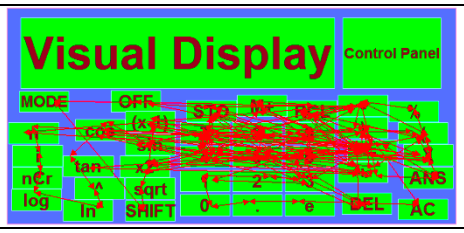
# APPENDIX L: TOP 5% 3 AREAS OPTIMISED DESIGN LAYOUTS WITH 10000 GENERATION SIZE

| Design Layout | ACT-R latency |
|---|---|
|  | 204.405 |
|  | 204.413 |
|  | 204.426 |
|  | 204.48 |
|  | 204.494 |
|  | 204.503 |
|  | 204.503 |

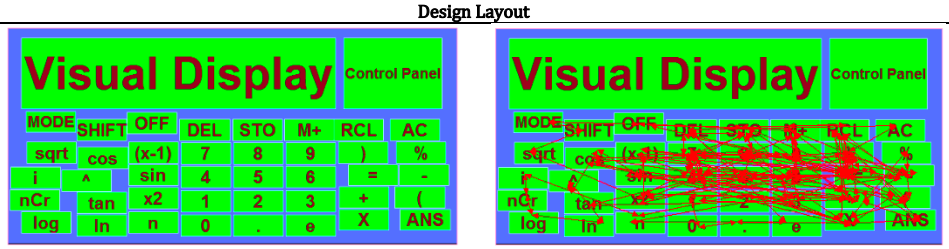
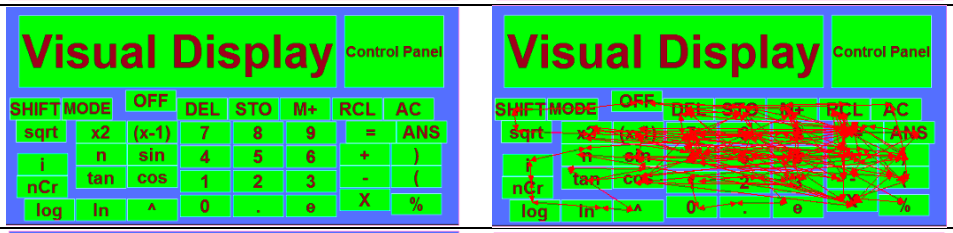
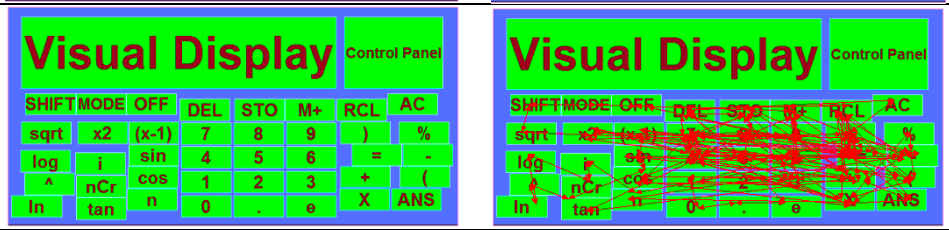
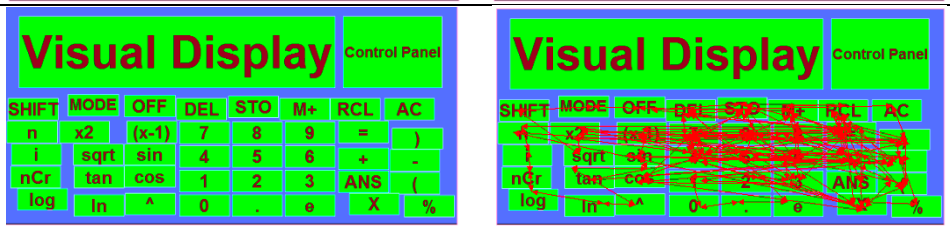
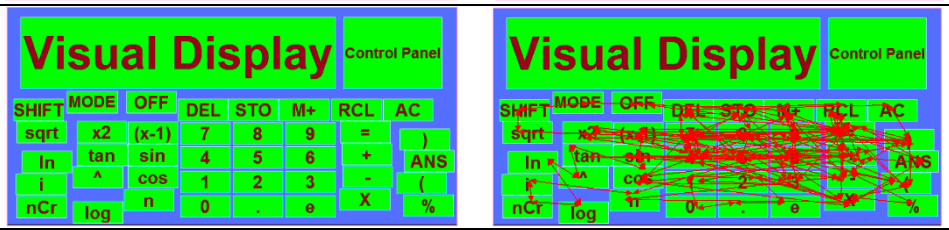
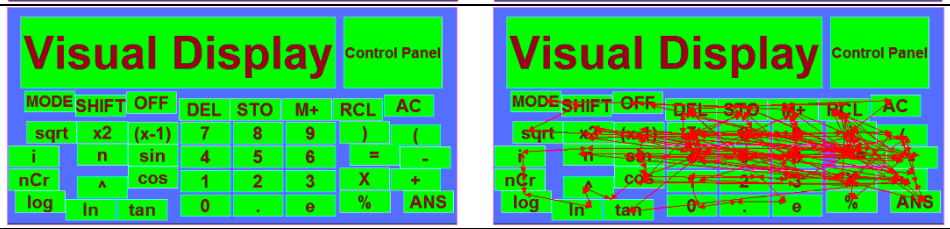
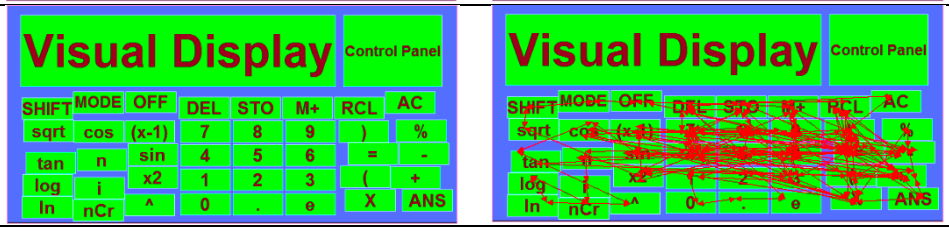
| | | |
|---|---|---|
|  |  | 204.523 |
|  |  | 204.524 |
|  |  | 204.531 |

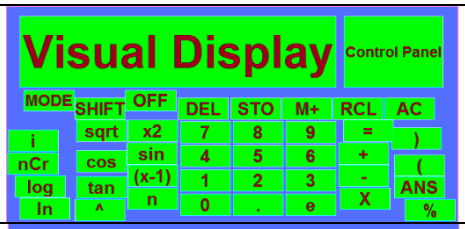
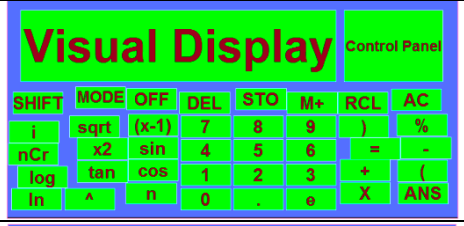
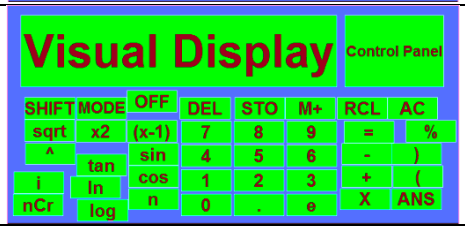
# APPENDIX M: TOP 5% 3 AREAS OPTIMISED DESIGN LAYOUTS WITH 20000 GENERATION SIZE

| Design Layout | | ACT-R latency |
|---|---|---|
|  |  | 204.408 |
|  |  | 204.457 |
|  |  | 204.461 |
|  |  | 204.48 |
|  |  | 204.483 |
|  |  | 204.484 |
|  |  | 204.487 |

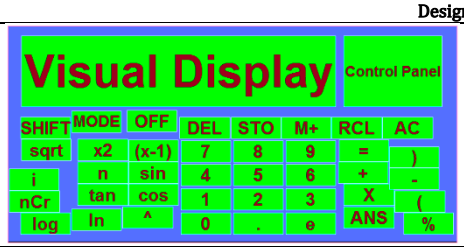
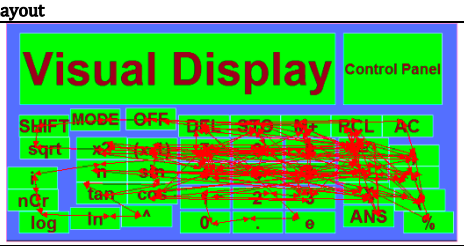
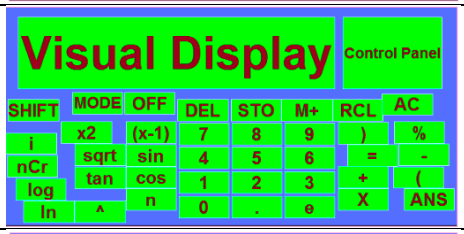
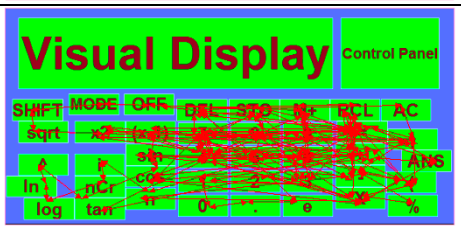
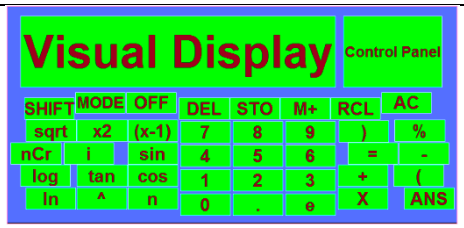
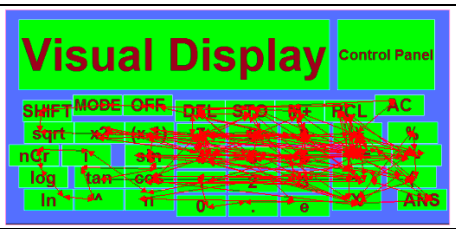
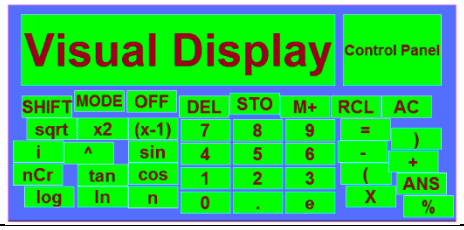
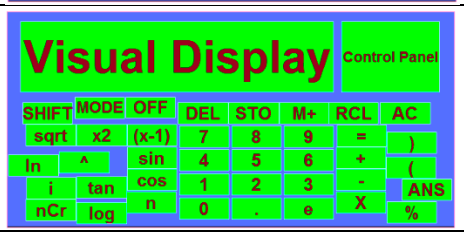
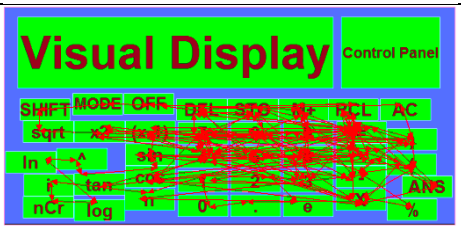
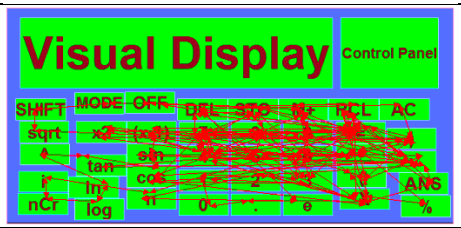
| | | |
|---|---|---|
|  |  | 204.494 |
|  |  | 204.501 |
|  |  | 204.511 |
|  |  | 204.512 |
|  |  | 204.514 |
|  |  | 204.514 |

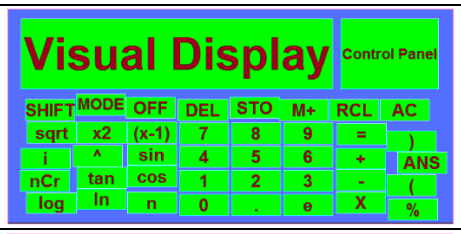
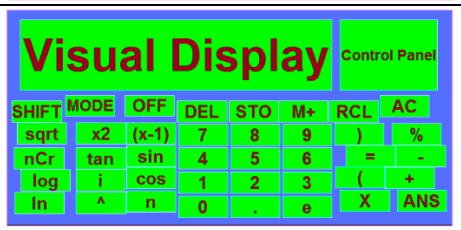
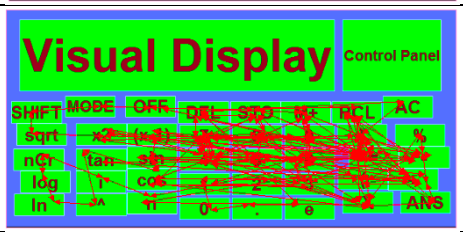
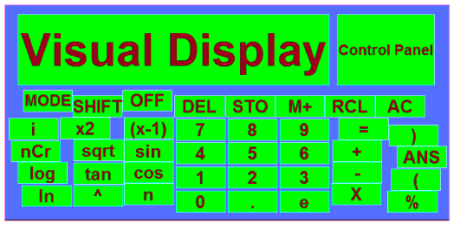
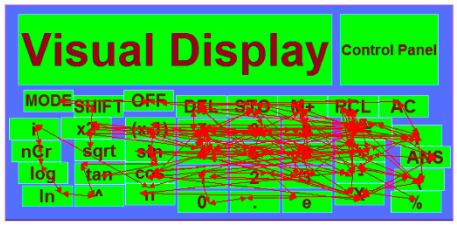
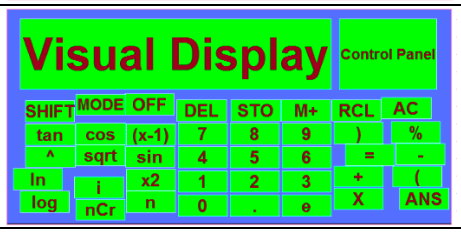
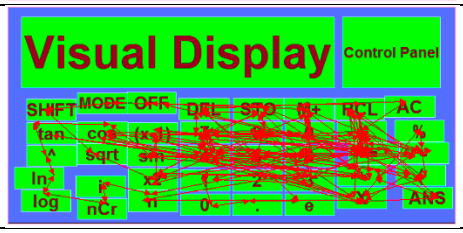
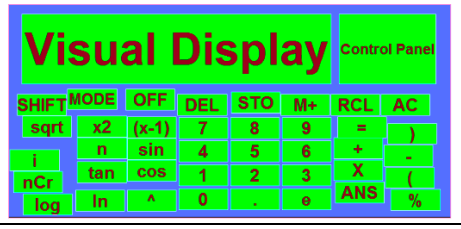
# APPENDIX N: TOP 5% 4 AREAS OPTIMISED DESIGN LAYOUTS WITH 5000 GENERATION SIZE

| Design Layout | ACT-R latency |
|---|---|
|  | 204.562 |
|  | 204.607 |
|  | 204.618 |
|  | 204.621 |
|  | 204.631 |
|  | 204.633 |
|  | 204.651 |

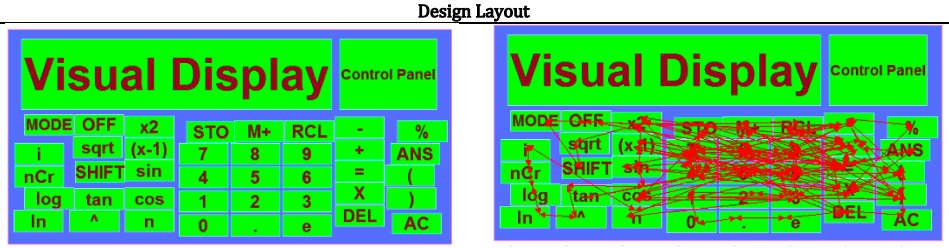
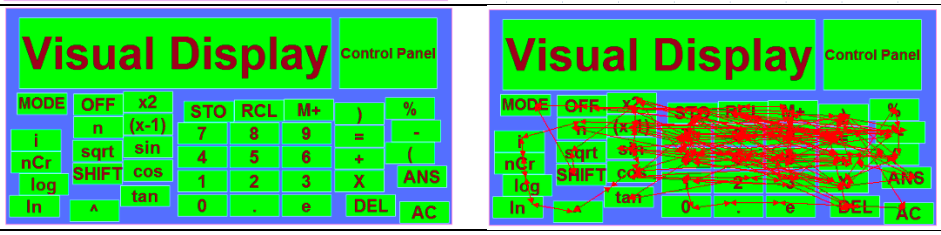
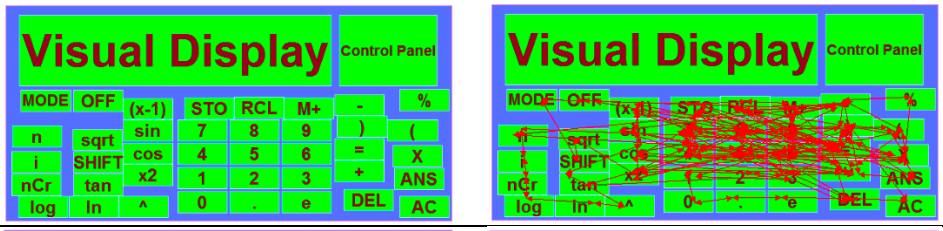
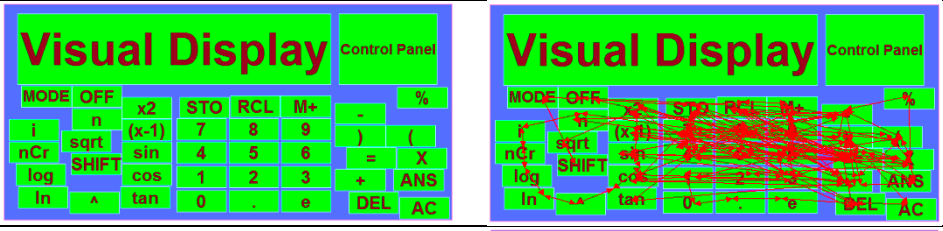
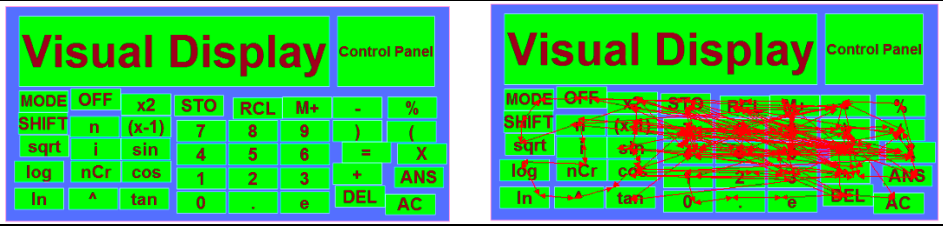
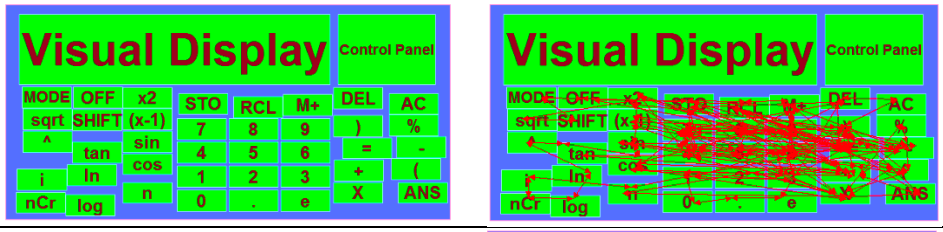
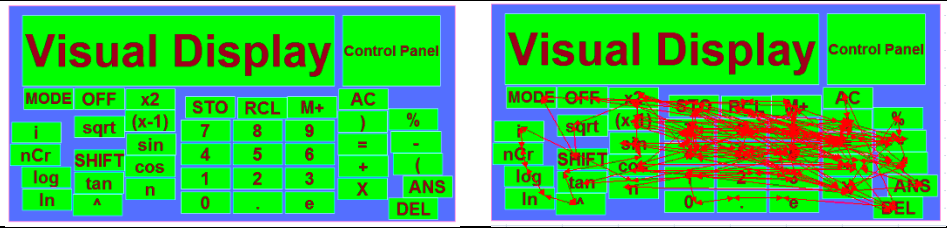
| | | |
|---|---|---|
|  |  | 204.659 |
|  |  | 204.660 |
|  |  | 204.662 |
|  |  | 204.678 |
|  |  | 204.679 |

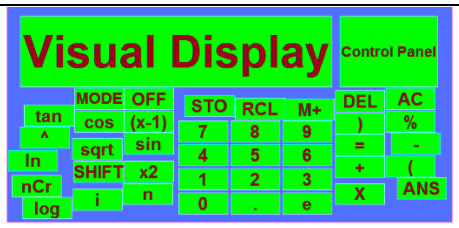
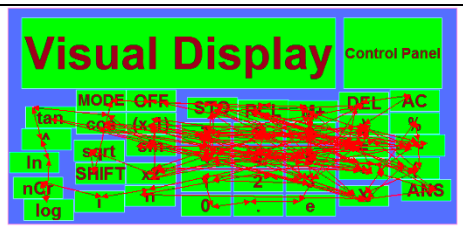
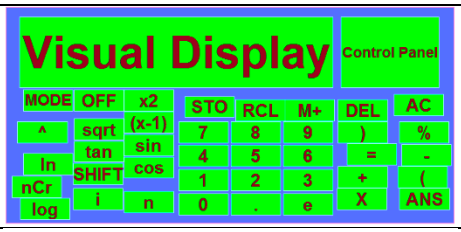
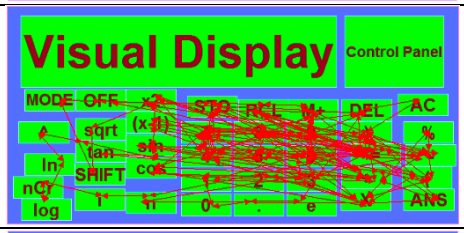
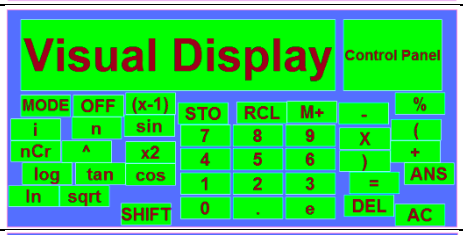
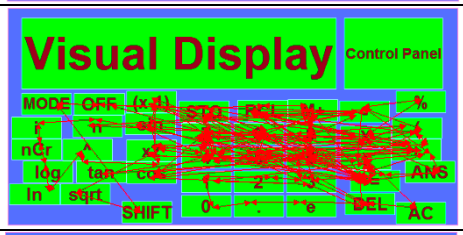
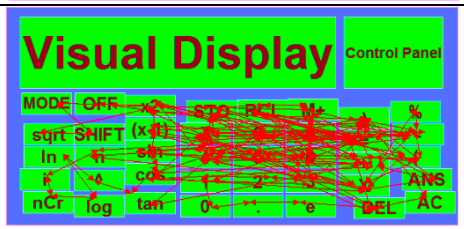
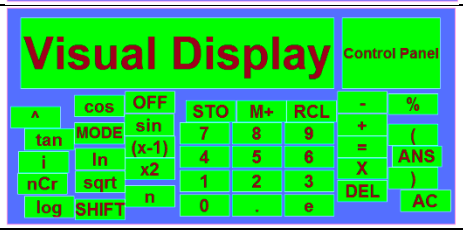
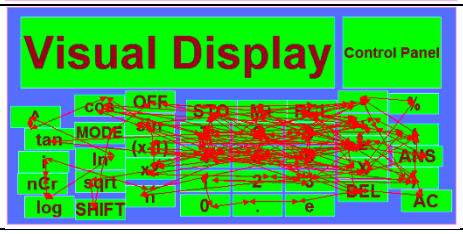
# APPENDIX O: TOP 5% 4 AREAS OPTIMISED DESIGN LAYOUTS WITH 20000 GENERATION SIZE

| Design Layout | | ACT-R latency |
|---|---|---|
|  |  | 204.441 |
|  |  | 204.451 |
|  |  | 204.470 |
|  |  | 204.495 |
|  |  | 204.497 |
|  |  | 204.500 |
|  |  | 204.500 |

| | | |
|---|---|---|
|  |  | 204.506 |
|  |  | 204.509 |
|  |  | 204.510 |
|  |  | 204.516 |
|  |  | 204.531 |

# APPENDIX P: TOP 5% 3 AREAS OPTIMISED DESIGN LAYOUTS WITH 5000 GENERATION SIZE

| Design Layout | ACT-R latency |
|---|---|
|  | 204.497 |
|  | 204.515 |
|  | 204.569 |
|  | 204.579 |
|  | 204.600 |
|  | 204.605 |
|  | 204.615 |

| | | |
|---|---|---|
|  |  | 204.616 |
|  |  | 204.621 |
|  |  | 204.629 |
|  |  | 204.630 |
|  |  | 204.632 |

# References


1. AbuBakar, 2010, Evaluation and Cognitive Modelling. Unpublished Master's Dissertation, The School of Computing Science & Digital Media, The Robert Gordon University,Aberdeen.
2. Adusumilli, Bein, D. & Bein, W., 2007. A Genetic Algorithm for the Two Machine Flow Shop Problem. p.8.
3. Ajenblit & Wainwright, 1998. Applying Genetic Algorithms to the U-Shaped Assembly Line Balancing Problem. Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98),,part of WCCI, Anchorage, Alaska, May 4-9, p.5.
4. Almasri, 2005. What We Can Learn From SILK and DENIM. School of Computer Science, McGill University, p.42.
5. Ames & Jie, 2004. Critical Paths for GUI Regression Testing. University of California, Santa Cruz.
6. Anderson, Matessa & Lebiere, 1997. ACT-R: A Theory of Higher Level Cognition and Its Relation to Visual Attention. Human-Computer Interaction, 12(4), pp.439–462.
7. Anderson, 1996. ACT: A simple theory of complex cognition. American Psychologist, 51(4), p.355.
8. Anderson, 1993. Rules of the mind, Lawrence Erlbaum.
9. Anderson & Lebiere, 1998. The atomic components of thought, Lawrence Erlbaum.
10. Anderson & Schunn, 2000. Implications of the ACT-R learning theory: No magic bullets. Advances in instructional psychology, 5, pp.1–34.
11. Anderson et al., 2004. An Integrated Theory of the Mind. Psychological Review, 111(4), pp.1036–1060.
12. Ando & Iba 2000. Analog Circuit Design with Variable Length Chromosomes-andoCEC2000. p.8.
13. Apple, 2012. Apple Development Centre Apple, ed. developer.apple.com. Available at: https://developer.apple.com/library/ios/ #documentation/UserExperience/ Conceptual/MobileHIG/Introduction/Introduction.html [Accessed 2012].
14. Balakrishnan, J. & Cheng, C.H., 2000. Genetic search and the dynamic layout problem. Computers and Operations Research, (27), pp.587–593.
15. Balaprakash, Wild & Hovland, 2012. An Experimental Study of Global and Local Search Algorithms in Empirical Performance Tuning. Proceedings of the 10th International Meeting on High- performance Computing for Computational Science (VECPAR 2012).
16. Balinsky, 2006. Evaluating interface aesthetics: measure of symmetry. Digital Publishing. Edited by Allebach, 6076, pp.52–63.
17. Bäck, 1989. Optimization by means of genetic algorithms. Technical University of Ilmenau, pp.163–169.
18. Bevan, 1998. ISO 9241: Ergonomic requirements for office work with visual display terminals (VDTs)-Part 11: Guidance on usability. TC, 159.

19. Bevan & Ferre, 2010. Usability Planner: Development of a Tool to Support the Selection and Estimation of Cost Benefits of UCD Methods. Submitted to HCSE.
20. Bielman, 2006. CFFI-SYS Interface Specification. pp.1–13.
21. Biswas, 2007. Simulating HCI for special needs. SIGACCESS Accessibility and Computing , Issue 89.
22. Biswas & Robinson, 2010. A brief survey on user modelling in HCI. Intelligent Techniques for Speech, Image and Language Processing, Springer-Verlag.
23. Biswas & Robinson, 2008. A Case Study of Simulating HCI for Special Needs. Designing Inclusive Futures, pp.175–184.
24. Biswas & Robinson, 2008. Automatic evaluation of assistive interfaces. In IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces. ACM Request Permissions.
25. Biswas & Robinson, 2007. Simulation to predict performance of assistive interfaces. Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility, pp.227–228.
26. Biswas, Bhattacharya & Samanta, 2005. User Model to Design Adaptable Interfaces for Motor-Impaired Users. Audio, Transactions of the IRE Professional Group on, pp.1–6.
27. Blouin & Fadel, 2002. GA-based multi-material structural optimization using stepwise mesh refinement. … Analysis and Optimization.
28. Bolton, Bass & Siminiceanu, R.I., 2012. Generating phenotypical erroneous human behavior to evaluate human–automation interaction using model checking. International Journal of Human-Computer Studies, pp.1–19.
29. Booch, 1986. Object-oriented development. IEEE Trans Software Eng.
30. Bothell, 2008. ACT-R 6.0 Reference Manual. pp.1–358.
31. Bowen, 2012. Windows Template Library. wtl.sourceforge.net. Available at: http://wtl.sourceforge.net/?source=navbar [Accessed 2012].
32. Brewbaker, 2008. Optimizing stylus keyboard layouts with a genetic algorithm: customization and internationalization. Dept. of Computer Science, Iowa State University.
33. Bryson, 2001. Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents.
34. Burk et al., 1998. Putting more genetics into genetic algorithms. Evolutionary Computation, 6(4), pp.387–410.
35. Byrne, 2003. A mechanism-based framework for predicting routine procedural errors. In Proceedings of the Twenty-Fifth Annual Conference of the Cognitive Science Society. Austin, TX.
36. Byrne, 2005. Cognitive architectures in HCI: Present work and future directions. Proceedings of the 11th International Conference on Human Computer Interaction, Las Vegas, NV.
37. Byrne et al., 1999. Eye tracking the visual search of click-down menus. Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit, pp.402–409.
38. Byrne, 2001. ACT-R/PM and menu selection: applying a cognitive architecture to HCI. International Journal of Human-Computer Studies, 55(1), pp.41–84.
39. Byrne, 2008. Cognitive Architecture. pp.1–21.

40. Byrne & Anderson, 2000. Serial Modules in Parallel: The Psychological Refractory Period and Perfect Time-Sharing.
41. Byrne & Kirlik, 2005. Using computational cognitive modeling to diagnose possible sources of aviation error. The International Journal of Aviation Psychology, 15(2), pp.135–155.
42. Card, Moran & Newell, 1980. The keystroke-level model for user performance time with interactive systems. Communications of the ACM, 23(7), pp.396–410.
43. Card, Moran & Newell, 1983. The psychology of human-computer interaction, CRC.
44. Carroll, 1997. Human-Computer Interaction: Psychology as a Science of Design. Annual review of psychology.
45. Carroll, 1988. Human–Computer Interaction. p.370. Available at: http://onlinelibrary.wiley.com/doi/10.1002/0470018860.s00545/full [Accessed March 16, 2011].
46. Carroll, 1999. Five Reasons for Scenario-Based Design. In HICSS '99: Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences. IEEE Computer Society.
47. Carroll, 2002. Scenarios and Design Cognition. In RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering. IEEE Computer Society.
48. Chesbrough & Rosenbloom, 2002. The role of the business model in capturing value from innovation: evidence from Xerox Corporation's technology spin-off companies. Industrial and Corporate Change.
49. Chong & Wray, 2006. Unified theories of cognition, Wiley Online Library.
50. Christou, 2007. CoDeIn: A Knowledge-based Framework for the Description and Evaluation of Reality-based Interaction,
51. Couckuyt et al., 2011. Automatic surrogate model type selection during the optimization of expensive black-box problems. pp.4269–4279.
52. Crystal & Ellington, 2004. Task analysis and human-computer interaction: approaches, techniques, and levels of analysis. Tenth Americas Conference on Information Systems.
53. Dahl, 2004. The birth of object orientation: The Simula languages. From Object-Orientation to Formal Methods.
54. Das & Stuerzlinger, 2007. A cognitive simulation model for novice text entry on cell phone keypads. In ECCE '07: Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore. ACM.
55. Dawson, 2010. Game Creation using OGRE – Object-Oriented Graphics Rendering Engine. pp.1–24.
56. Deininger, 1960. Human factors engineering studies of the design and use of pushbutton telephone sets. Bell System Technical Journal, 39, pp.995–1012.
57. Dillon, 2005. Encyclopedia of Cognitive Science. p.16.
58. Dillon, Sweeney & Maguire, 1994. A survey of usability engineering within the European IT industry - Current practice and needs. People and computers VIII: proceedings of HCI 93, Loughborough, September 1993, (7), p.81.
59. Dorman & Gaudiano, 1998. Motivation. In Handbook of Brain Theory and Neural Networks. M. Arbib (Ed.). Cam- bridge, MA: MIT Press., pp. 1–15.

60. Dufilie et al., 2012. Weave: A Web-based Architecture Supporting Asynchronous and Real-time Collaboration.
61. Fahimuddin, 2003. PLATON Optimisation Using GALib.
62. Fleetwood & Byrne, 2001. Modeling Icon Search in ACT-R/PM. ICCM 2001.
63. Fleetwood & Byrne, 2006. Modeling the visual search of displays: a revised ACT-R model of icon search based on eye-tracking data. Human-Computer Interaction, 21(2), pp.153–197.
64. Fleetwood & Byrne, 2003. Modeling the visual search of displays: A revised ACT-R/PM model of icon search based on eye-tracking and experimental data. Proceedings of the Fifth International Conference on Cognitive Modeling, pp.87–92.
65. Fleetwood et al., 2006. Putting the Brain in the Box for Human-System Interface Evaluation. Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 50(11), pp.1165–1169.
66. Fogel, 1994. An introduction to simulated evolutionary optimization. Neural Networks, IEEE Transactions on, 5(1), pp.3–14.
67. Forrest & Mitchell, 1993. What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. Machine Learning, 13(2), pp.285–319.
68. Friedman, 2009. Network-on-chip placement & Routing Overview. p.15.
69. Fu, 2001. The influence of experienced effort on learning and choice solution paths in a simple navigation task. In Fourth International Conference on Cognitive Modeling. pp. 279–280.
70. Fu, 2003. Adaptive Planning in Problem Solving-Cost-Benefit Tradeoffs Under Bounded Rationality. University of Hong Kong.
71. Fu & Pirolli 2007. SNIF-ACT: A cognitive model of user navigation on the World Wide Web. Human-Computer Interaction, 22(4), pp.355–412.
72. Gamsriegler, 2006. Reasons for success of usability measures in small-sized software development companies.
73. Gary & Adrian, 2008. Learning OpenCV.
74. Gero, 1999. A Genetic Programming Approach To The Space Layout Planning Problem. in R. Junge (ed.), CAAD Futures 1997, Kluwer, Dordrecht, pp.875–884.
75. Gersh, McKneely & Remington, 2005. Cognitive engineering: Understanding human interaction with complex systems. Johns Hopkins APL technical digest, 26(4), pp.377–382.
76. Ginbayashi, 2004. Latest progress and trends in Java/EJB technologies. FUJITSU Sci Tech J.
77. Goldberg & Deb, 1991. A comparative analysis of selection schemes used in genetic algorithms. Foundations of genetic algorithms, 1, pp.69–93.
78. Golovine, McCall & O'Brian Holt, 2010. Evolving interface designs to minimize user task times as simulated in a cognitive architecture. Evolutionary Computation (CEC), 2010 IEEE Congress on, pp.1–7.
79. Gray & Altman, 1999. Cognitive modeling and human-computer interaction. International encyclopedia of ergonomics and human factors. New York: Taylor & Francis, Ltd.

80. Gray, John & Atwood, 1993. Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance. Human-Computer Interaction, 8(3), pp.237–309.
81. Gregori, 2012. Introduction to Computer Vision using OpenCV. In 2012 Embedded Systems Conference in San Jose. pp. 1–34.
82. Grosz, 2005. Beyond mice and menus. Proceedings of the American Philosophical Society, 149(4), pp.529–543.
83. Grudin, 2005. Three faces of human-computer interaction. Annals of the History of Computing, IEEE, 27(4), pp.46–62.
84. Haapasalo & Kess, 2002. Usability evaluation of user interfaces in complex tasks–Does CAD UI apply to practical work in architecture? In Twenty-Sixth Annual International Computer Software and Application Conference (COMPSAC 2002).
85. Halbrugge et al., 2007. ACT-CV: Die Erweiterung von ACT-R um Bildverarbeitungsalgorithmen erlaubt die schnelle Erzeugung mächtiger Benutzermodelle. Simulationsgestützte Systemgestaltung - DGLR-Bericht, 04, pp.313–331.
86. Halbrugge, Deml & Neumann, 2008. A Cognitive Model of Visual Search on Direction Signs alongside the Autobahn.
87. Hancock, 1994. An empirical comparison of selection methods in evolutionary algorithms. Evolutionary Computing, pp.80–94.
88. Harik, Lobo & Goldberg, 1999. The Compact Genetic Algorithm. IEEE Transactions on Evolutionary Computation, 3(4), pp.523–528.
89. Hartwig, Darolti & Herczeg, 2003. Lightweight Usability Engineering Scaling Usability-Evaluation to a Minimum? Human Computer Interaction-Theory and Practice (Part I), London: Lawrence Erlbaum Associates Publishers, pp.474–478.
90. Haupt & Haupt, 2004. Practical Genetic Algorithms 2nd ed, Wiley-Blackwell.
91. Ho & Perng, 2009. An Application of Genetic Simulation Approach to Layout Problem in Robot Arm Assembly Factory. p.7.
92. Holland, 1975. Adaptation in Natural and Artificial Systems, The University of Michigan Press.
93. Holzhamper & Seppelt, 2007. A generic tool for optimising land-use patterns and landscape structures. Environmental Modelling & Software 22 (2007) 1801e1804, p.4.
94. Hooda, 2012. An Automation of Software Testing: A Foundation for the Future. International Journal of Latest Research in Science and Technology, 1(2), pp.152–154.
95. Huang, Breheny & Ma, 2012. A Selective Review of Group Selection in High Dimensional Models. Audio, Transactions of the IRE Professional Group on.
96. Intel, 2010. open source computer vision library, Intel Corporation,
97. Irby et al., 1977. A methodology for user interface design. Systems Development, Xerox.
98. Ismail, Abiddin & Hassan, 2011. Improving the Development of Postgraduates' Research and Supervision. International Education Studies, 4(1), p.p78.
99. Ivory & Hearst, 1999. Automated usability evaluation of WIMP and Web interfaces. Submitted for publication.
100. Ivory & Hearst, 2001. The state of the art in automating usability evaluation of user interfaces. ACM Computing Surveys (CSUR), 33(4), pp.470–516.

101. Jadaan, Rajamani & Rao, 2008. Improved selection operator for ga. Journal of Theoretical and Applied Information Technology, 4(4), p.269277.
102. Jagielski & Gero, 2009. A Genetic Programming Approach to the Space Layout Planning Problem. CAAD Futures, 875, pp.875–884.
103. Jang, Kim & Russell, 2003. Manage Space for Construction Facilities on High-rise Buildings. Proceedings of Construction Research Congress: Winds of Change: Integration and Innovation In Construction, ASCE Construction Congress VII, Honolulu, HI, pp.19–21.
104. Jin, 2011. Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation, 1(2), pp.61–70.
105. John et al., 2004. Predictive human performance modelling made easy. Proceedings of the SIGCHI conference on Human factors in computing systems, pp.455–462.
106. John, 2008. CogTool User Guide,
107. John, 2010. Reducing the variability between novice modelers: Results of a tool for human performance modeling produced through Human-Centered Design. Proceedings of the 19th Annual Conference on Behavior Representation in Modeling and Simulation (BRIMS), pp.22–25.
108. John & Kieras, 1994. The GOMS Family of Analysis Techniques: Tools for Design and Evaluation. The GOMS Family of Analysis Techniques: Tools for Design and Evaluation.
109. John & Kieras, 1996. The GOMS family of user interface analysis techniques: comparison and contrast. Transactions on Computer-Human Interaction (TOCHI, 3(4).
110. John & Kieras, 1996. Using GOMS for user interface design and evaluation: which technique? ACM Transactions on Computer-Human Interaction (TOCHI), 3(4), pp.287–319.
111. John & Prevas, 2003. New tools for modeling for HCI evaluation. p.3.
112. John & Salvucci, 2005. Multipurpose prototypes for assessing user interfaces in pervasive computing systems. Pervasive Computing, IEEE, 4(4), pp.27–34.
113. Johnson et al., 2002. A Model of Spatio-Temporal Coding of Memory for Multidimensional Stimuli. 24th Annual Meeting of the Cognitive Science Society, p.6.
114. K Card, P Moran & Newell, 1983. The psychology of human-computer interaction. In The psychology of human-computer interaction. CRC, p. 488.
115. Kahne, 1997. A genetic algorithm-based place-and-route compiler for a run-time reconfigurable computing system. Virginia Polytechnic Institute and State University.
116. Karat, 1993. Usability engineering in dollars and cents. Software, IEEE, 10(3), pp.88–89.
117. Karray et al., 2008. Human-computer interaction: Overview on state of the art. International Journal on Smart Sensing and Intelligent Systems, 1(1), pp.137–159.
118. Kay, 1972. A personal computer for children of all ages. In Proceedings of the ACM national conference.
119. Kay, 1996. The Early History of Smalltalk. History of Programming Languages II, p.535.

120. Kay & Goldberg, 1977. Personal dynamic media. Computer, 10(3), pp.31–41.
121. Kieras, 1996. A Guide to GOMS Model Usability Evaluation using NGOMSL. ukpmcacuk.
122. Kieras, 1997. Task analysis and the design of functionality. Handbook of computer science and engineering, CRC Press, Inc.
123. Kieras, 1999. A guide to GOMS model usability evaluation using GOMSL and GLEAN3. Ann Arbor, MI.
124. Kieras, 2005. Computational Modeling of Human Multiple-Task Performance.
125. Kieras, 2006. A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN4. 2006 [cited September 20th, 2006].
126. Kieras, 2011. Extending and Applying the EPIC Architecture for Human Cognition and Performance, DTIC Document.
127. Kieras & Meyer, 1994. The EPIC architecture for modeling human information-processing and performance: A brief introduction.
128. Kieras & Meyer, 1997. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. Human-Computer Interaction, 12(4), pp.391–438.
129. Kieras & Meyer, 2004. The EPIC architecture: Principles of operation,
130. Kieras, Wood & Meyer, 1997. Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. ACM Transactions on Computer-Human Interaction (TOCHI), 4(3), pp.230–275.
131. Kindler, 2007. Simulation and 40 Years of Object-Oriented Programming.
132. Kirkpatrick, Gelatt & Vecchi, 1983. Optimization by Simulated Annealing. Science, 220(4), pp.671–680.
133. Klanac et al., 2008. Structural omni-optimization of a tanker. 7th International Conference on Computer and IT Applications in the Maritime Industries, COMPIT, 8, pp.537–550.
134. Kljajevic, 2008. Cognitive Models as Usability Testing Tools. igi-globalcom.
135. Koehn, 1994. Combining Genetic Algorithms and Neural Networks: The Encoding Problem. p.67.
136. Konak, Coit & Smith, 2006. Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety, 91(9), pp.992–1007.
137. Lacevic, Konjicija & Avdagic, 2007. Population diversity measure based on singular values of the distance matrix. Audio, Transactions of the IRE Professional Group on, pp.1863–1869.
138. Laird, 2012. The Soar Cognitive Architecture, The MIT Press, 55 Hayward Street, Cambridge, MA 02142-1315 USA.
139. Laird & Congdon, 1998. The Soar User's Manual Version 9.3.
140. Laird & Rosenbloom, 1992. In pursuit of mind: the research of Allen Newell. AI Magazine, 13(4), p.17.
141. Landay, 1996. SILK: sketching interfaces like krazy. Conference companion on Human factors in computing systems: common ground, pp.398–399.
142. Landay & Myers, 1993. Extending an existing user interface toolkit to support gesture recognition. INTERACT"93 and CHI"93 conference companion on Human factors in computing systems, pp.91–92.

143. Lane, 2007. The power of the bond between cause and effect: Jay Wright Forrester and the field of system dynamics. System dynamics review, 23(2-3), pp.95–118.
144. Langtangen, 2009. Python Scripting for Computational Science. Python Scripting for Computational Science, 3rd edition.
145. Lebiere, 2006. ACT. Encyclopaedia of Cognitive Science.
146. Lebiere et al., 2002. IMPRINT/ACT-R: Integration of a task network modeling architecture with a cognitive architecture and its application to human error modeling. Simulation Series, 34(3), pp.13–18.
147. Lebiere et al., 2005. Integrating Modeling and Simulation into a General-Purpose Tool. human-factors.arc.nasa.gov.
148. Lewis & Rieman, 1993a. Task Centered User Interface Design: A Practical Introduction. University of Colorado, Boulder. This is shareware book that is available online at the following url: ftp://ftp. cs. colorado. edu/pub/cs/distribs/clewis/HCI-Design-Book.
149. Lidwell, Holden, & Buttler, 2003. Universal Principles of Design 1st ed, Rockport Publishers Inc. Available at: http://www.qbookshop.com/products/153021/9781592530076/Universal-Principles-of-Design.html.
150. Lif et al., 2003. A Structural Approach to Prototype Design in Iterative System Development,
151. Lim et al., 2007. A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation. pp.1288–1295.
152. Lin et al., 1999. DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site. DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site.
153. Ling, Samsudin & Abduliah, 2000. Aesthetic measures for assessing graphic screen. Journal of Information Science and Engineering, 16(97), p.116.
154. Lohn, Larchev & DeMara, 2002. A Genetic Representation for Evolutionary Fault Recovery in FPGAs. International Conf. on Military and Aerospace Programmable Logic Devices, Laurel, MD.
155. Luke & Spector, 2010. A Comparison of Crossover and Mutation in Genetic Programming. p.9.
156. Luke & Spector, 2010. A Revised Comparison of Crossover and Mutation in Genetic Programming. p.8.
157. Luo & Siewiorek, 2007. KLEM: A Method for Predicting User Interaction Time and System Energy Consumption during Application Design. In ISWC '07: Proceedings of the 2007 11th IEEE International Symposium on Wearable Computers. IEEE Computer Society, pp. 69–76.
158. Machover, 1994. Four decades of computer graphics. Computer Graphics and Applications, IEEE, 14(6), pp.14–19.
159. Macleod, 1994. Usability: Practical methods for testing and Improvement. Proceedings of the Norwegian Computer Society Software, 94, p.4.
160. Manikas & Cain, 1996. Genetic algorithms vs. simulated annealing: A comparison of approaches for solving the circuit partitioning problem. Pittsburgh (PA): University of Pittsburgh.
161. Maveryx, 2011. Maveryx User Manual, Maveryx.

162. McGrenere, 1998. Learning to use complex computer technology: The importance of user interface design. PhD Depth Paper, Department of Computer Science, University of Toronto.
163. Metropolis et al., 1953. Equation of state calculations by fast computing machines. The journal of chemical physics, 21, p.1087.
164. Miller & Goldberg, 1995. Genetic Algorithms, Tournament Selection, and the Effects of Noise. Complex Systems, 9, pp.193–212.
165. Miller & Goldberg, 1996. Genetic algorithms, selection schemes, and the varying effects of noise. Evol Comput. MIT Press. Cambridge, MA, USA, 4(2), pp.113–131.
166. Mitchell, 1996. An introduction to genetic algorithms. An introduction to genetic algorithms.
167. Mitchell & Forrest, 1994. Genetic algorithms and artificial life. Artificial Life, 1(3), pp.267–289.
168. Myers, 1993. Why are human-computer interfaces difficult to design and implement?
169. Narahara & Terzidis, 2007. Multiple-constraint Genetic Algorithm in Housing Design. p.8.
170. Newell, 1990. Unified Theories of Cognition, Harvard University Press, Cambridge, Massachusetts.
171. Newman et al., 2003. DENIM: An informal web site design tool inspired by observations of practice. Human-Computer Interaction, 18(3), pp.259–324.
172. Newman, 1968. A system for interactive graphical programming. In AFIPS Spring Joint Computing Conference. pp. 47–54.
173. Nielsen, 1994. Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier. useit. com, Papers and essays. Acedido em 12 de Julho de 2005.
174. Nolle et al., 2002. Simulated annealing and genetic algorithms applied to finishing mill optimisation for hot rolling of wide steel strip. International Journal of Knowledge Based Intelligent Engineering Systems, 6(2), pp.104–111.
175. Nygaard & Dahl, 1981. The development of the SIMULA languages R. L. Wexelblat, ed., ACM.
176. Oliver, Monmarche & Venturini, 2002. Interactive design of web sites with a genetic algorithm. Proceedings of the IADIS International Conference www/Internet, pp.355–362.
177. Olson & Olson, 2003. Human-Computer Interaction: Psychological Aspects of the Human Use of Computing. Annual review of psychology, 54(1), pp.491–516.
178. OpenCV, 2012. OpenCV. opencv.org. Available at: http://opencv.org/ [Accessed 2012].
179. Osman, Georgy & Ibrahim, 2003. A hybrid CAD-based construction site layout planning system using genetic algorithms. Automation in Construction, 12, pp.749 – 764.
180. Pang & Schauder, 2007. The culture of information systems in knowledge-creating contexts: The Role of user-centred design. Informing Science: International Journal of an Emerging Transdiscipline, 10, pp.203–235.
181. Peebles & Bothell, 2004. Modelling performance in the Sustained Attention to Response Task. p.6.

182. Perrins, 2008. Introduction to the Dojo Toolkit. In 2008 IBM Corporation. IBM Software Group Lab Services. pp. 1–40.
183. Petrovic, 2001. Solving LEGO brick layout problem using Evolutionary Algorithms. Informatikkonferanse NIK'2001, p.11.
184. Pérez et al., 2005. Optimal Layout Design for Milk Goats Livestock Farms Using Genetic Algorithms. p.17.
185. Purvis, 2009. A Genetic Algorithm Approach to Automated Custom Document Assembly. p.6.
186. Rajanen, 2011. Applying usability cost-benefit analysis - explorations in commercial and open source software development contexts,
187. Rajanen & Jokela, 2007. Analysis of Usability Cost-Benefit Models.
188. Redmond & Smith, 1975. Project Whirlwind,
189. Reeves, 2003. Genetic algorithms, Springer.
190. Ride, 2012. ER1: Personal Robot System. Evolution Robotics. Available at: http://evo.stage.la.idealab.com/er1/ [Accessed 2012].
191. Rieman, Young & Howes, 1996. A dual-space model of iteratively deepening exploratory learning. International Journal of Human-Computer Studies, 44(6), pp.743–775.
192. Ritter, 2003. Techniques for modeling human performance in synthetic environments, Human Systems Information Analysis Center.
193. Ritter & Young, 2001. Embodied models as simulated users: Introduction to this special issue on using cognitive models to improve interface design. International Journal of Human-Computer Studies, 55(1), pp.1–14.
194. Ritter et al., 2006. Providing user models direct access to interfaces: an exploratory study of a simple interface with implications for HRI and HCI. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 36(3), pp.592–601.
195. Ritter, Jones & Baxter, 1998. Reusable models and graphical interfaces: Realising the potential of a unified theory of cognition. Mind modeling-A cognitive science approach to reasoning, learning and discovery, pp.83–109.
196. Ritter, Kukreja & St-Amant, 2007. Including a model of visual processing with a cognitive architecture to model a simple teleoperation task. Journal of Cognitive Engineering and Decision Making, 1(2), pp.121–147.
197. Ritter, van Rooy & St-Amant, 2002. A user modeling design tool based on a cognitive architecture for comparing interfaces. Computer-Aided Design of User Interfaces III, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI, pp.111–118.
198. Rosson & Carroll, 2009. Scenario based design. Human-computer interaction. Boca Raton, FL, pp.145–162.
199. Roth, Patterson & Mumaw, 2002. Cognitive engineering: issues in user-centered system design. Encyclopedia of software engineering, pp.163–179.
200. Salvucci, 2000. A Model of Eye Movements and Visual Attention.
201. Salvucci, 2001. An integrated model of eye movements and visual encoding. Cognitive Systems Research, 1(4), pp.201–220.
202. Salvucci, 2001. Predicting the effects of in-car interface use on driver performance: An integrated model approach. International Journal of Human-Computer Studies, 55(1), pp.85–107.

203. Salvucci, 2001. Predicting the effects of in-car interfaces on driver behavior using a cognitive architecture. Proceedings of the SIGCHI conference on Human factors in computing systems, pp.120–127.
204. Salvucci, 2001. Predicting the effects of in-car interface use on driver performance: An integrated model approach. International Journal of Human-Computer Studies, 55(1), pp.85–107.
205. Salvucci, 2002. Modeling driver distraction from cognitive tasks. Proceedings of the 24th Annual Conference of the Cognitive Science Society, pp.792–797.
206. Salvucci & Lee, 2003. Simple cognitive modeling in a complex cognitive architecture. CHI 2003. ACM Conference on Human Factors in Computing Systems, CHI Letters, 5(1), pp.265–272.
207. Salvucci, 2005. A multitasking general executive for compound continuous tasks. Cognitive Science, 29(3), pp.457–492.
208. Salvucci, Boer & Liu, 2001. Toward an integrated model of driver behavior in cognitive architecture. Transportation Research Record: Journal of the Transportation Research Board, 1779(-1), pp.9–16.
209. Salvucci et al., 2005. Distract-R: Rapid prototyping and evaluation of in-vehicle interfaces. Proceedings of the SIGCHI conference on Human factors in computing systems, pp.581–589.
210. Salvucci & Macuga, 2002. Predicting the effects of cellular-phone dialing on driver performance. Cognitive Systems Research, 3(1), pp.95–102.
211. Salvucci, Taatgen & Kushleyeva, 2006. Learning when to switch tasks in a dynamic multitasking environment. Proceedings of the seventh international conference on cognitive modeling, pp.268–273.
212. Schultheis, Barkowsky & Bertel, 2006. LTMC - An Improved Long-Term Memory for Cognitive Architectures. p.6.
213. Shah et al., 2003. Connecting a cognitive model to dynamic gaming environments: Architectural and image processing issues. Proceedings of the Fifth International Conference on Cognitive Modeling (ICCM), pp.189–194.
214. Shoup, 2001. SuperPaint: An early frame buffer graphics system. Annals of the History of Computing.
215. Sivaraj & Ravichandran, 2011. A review of selection methods in genetic algorithm. International Journal of Engineering Science and Technology, 3(5), pp.3792–3797.
216. Smith, 1975. Pygmalion: a creative programming environment., p.243.
217. Smith et al., 1986. The star user interface: an overview. pp.383–396. Available at: http://www.worldcat.org/title/national-computer-conference-june-16-19-1986-las-vegau-nevada/oclc/255548197 [Accessed March 19, 2011].
218. Sonnenburg et al., 2010. The SHOGUN machine learning toolbox. The Journal of Machine Learning Research, 11, pp.1799–1802.
219. Spears, 1993. Crossover or Mutation. p.17.
220. St Amant & Ritter, 2004. Automated GOMS–to–ACT-R model generation. Proceedings of the Sixth International Conference on Cognitive Modeling.
221. St Amant et al., 2005. Image processing in cognitive models with SegMan. Proceedings of the 11th International Conference on Human-Computer Interaction, HCII, 2005.

222. St Amant, Horton & Ritter, 2004. Model-based evaluation of cell phone menu interaction. Proceedings of the SIGCHI conference on Human factors in computing systems, pp.343–350.
223. St-Amant, Freed & Ritter, 2005. Specifying ACT-R models of user interaction with a GOMS language. Cognitive Systems Research.
224. St Amant, McBride & Ritter, 2006. AI support for building cognitive models. In Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI), Nectar Track, Menlo Park.
225. Stewart & West, 2006. Deconstructing ACT-R. Proceedings of the seventh international conference on cognitive modeling, pp.298–303.
226. Stone et al., 2005. User Interface Design and Evaluation (Interactive Technologies), Morgan Kaufmann.
227. Sutherland, 1964. Sketch pad a man-machine graphical communication system. In Proceedings of the SHARE design automation workshop. pp. 6.329–6.346.
228. SWIG, 2012. SWIG Documentation. Pp.1–731.
229. Taatgen, 1996. A model of free-recall using the ACT-R. Benelearn, p.10.
230. Tate & Smith, 1998. Expected Allele Coverage and the Role of Mutation in Genetic Algorithms. p.7.
231. Teo & John, 2006. Comparisons of keystroke-level model predictions to observed data. CHI “06: CHI ”06 extended abstracts on Human factors in computing systems.
232. Teo & John, 2011. The Evolution of a Goal-Directed Exploration Model: Effects of Information Scent and GoBack Utility on Successful Exploration. Topics in Cognitive Science, 3(1), pp.154–165.
233. Teo, John & Pirolli, 2007. Towards a tool for predicting user exploration. CHI “07: CHI ”07 extended abstracts on Human factors in computing systems.
234. Thapatsuwan, Chainate & Pongcharoen, 2007. Improving Packing Efficiency for Shipping Container. International Journal of the Computer, the Internet and Management, 15(4), pp.24.1–6.
235. Thierens, 2002. Adaptive mutation rate control schemes in genetic algorithms. Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on, 1, pp.980–985.
236. Thompson & Stevenson, 2011. Mobile Augmented Reality for Grocery Shopping. Symposium (MICS) Conference, Duluth, MN.
237. TNO, 2012. TNO. Usability Testing .nl. Available at: http://www.usabilitytesting.nl/ usability_test.html [Accessed 2012].
238. Troutt & Sheiner, 2007. Human-Computer Interface Design and Usability Testing. pp.1–20.
239. Van Rooy, Ritter & St-Amant, 2002. Using a simulated user to explore human robot interfaces. Article under review. Draft at http://www4. ncsu. edu/stamant/papers/ HRI-v4_06-25. pdf.
240. Vera et al., 2005. Automating human-performance modeling at the millisecond level. Human-Computer Interaction, 20(3), pp.225–265.
241. Verhaegh, Aarts & Korst, 2006. Intelligent Algorithms in Ambient and Biomedical Computing, Dordrecht: Springer.
242. Wall, 1996. A genetic algorithm for resource-constrained scheduling.
243. Wall, 1998. GAlib: A C++ Library of Genetic Algorithm Components. p.104.

244. Wall, 2011. Abbot framework for automated testing of Java GUI components and programs. abbot.sourceforge.net. Available at: http://abbot.sourceforge.net/doc/overview.shtml [Accessed 2011].
245. Woods & Roth, 1988. Cognitive engineering: human problem solving with tools. Human Factors, 30(4).
246. Yang, Shan & Yeh, 2002. The Study of Optimal Slurry Wall Unit Layout by Genetic Algorithm. The Ninth International Conference on Computing in Civil and Building Engineering, p.6.
247. Zachary et al., 2001. 5. The application of human modeling technology to the design, evaluation and operation of complex systems. In Emerald Group Publishing Limited.
248. Zhai, 2004. The Computer Mouse and Related Input Devices. In Berkshire Encyclopedia of Human-Computer Interaction. W.S. Bainbridge (ed).
249. Zhou et al., 2007. Combining Global and Local Surrogate Models to Accelerate Evolutionary Optimization. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 37(1), pp.66–76.
250. Zhu, 2003. Population Diversity in Genetic Algorithm for Vehicle Routing Problem with Time Windows. Department of Computer Science, National University of Singapore.
251. Zitzler, Deb & Thiele 2000. Comparison of multiobjective evolutionary algorithms: empirical results. Evolutionary Computation, 8(2), pp.173–195.