ISSN 0254–2757

QI QUÆSTIONES INFORMATICÆ

Volume 6 • Number 1 May 1988

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereeniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

The annual subscription is

	SA	US	UK
Individuals	R20	$ 7	£ 5
Institutions	R30	$14	£10

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Wirth / Hoare / Dijkstra / Gries / Bach

Kürchen

Kowalski

Liwette

Guest Editorial

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial

Volume six of QI heralds several changes. The most visible is the change in format. The black on red cover has been changed to a more readable blue on white, but we have retained the style of the old cover, for the sake of continuity. The papers are now set in a tighter format, using double columns, which will enable more papers to be published for the same cost.

For authors, the most significant change is that as from Volume 6 Number 2 (the next issue), a charge will be made for typesetting. The charge is quite modest – R20 per page – and will enable us to keep up the high standards that we have become used to with QI. It is worth recording that the alternative to this suggestion was that authors should present camera-ready typescript, as is done for *Quæstiones Mathematicæ*. Given that document preparation and electronic typesetting is one of the areas of computer science that we can feel proud of, it seemed right that our journal should use the most modern techniques available. Fortunately, the two controlling bodies, the CSSA and SAICS, eventually agreed to our proposal and the result is the professional journal you have in front of you now.

Supporters of QI may be interested in a few statistics that I compiled when I took over the editorship from Gerrit Wiechers in April this year. In the past two years (June 1985 to June 1988), 73 papers have been received. Of these 39 (53%) have appeared, 19 have been rejected or withdrawn (26%) and 15 (21%) are either with authors for changes or with referees. If we look at the complete picture for Volumes 4 and 5, we find the following:

Volume	**Issues**	**Papers**	**Pages**	**Ave. pages per paper**
5	3	27*	220	7.7
4	3	21	136	6.4

Although this issue contains one very long paper of 18 pages, the future policy of QI will be to restrict papers to 6 or 7 printed pages, and prospective authors are asked to bear this in mind when submitting papers.

For the future, we are hoping to move towards more special issues. Many of the papers being published at the moment were presented at the 4th SA Computer Symposium in 1987. Instead of continuing the policy of allowing such papers to be accepted by QI without further refereeing, we are hoping to negotiate with Conference organisers to produce special issues of QI. Thus the proceedings would *ab initio* be typeset by QI and all the papers would be in a single issue. Given the competitive charges of QI, there will be financial gains for both parties in such an arrangement.

As this is my first editorial, it is fitting that it should close with a tribute to the previous QI team. My predecessor as editor was Gerrit Wiechers. Gerrit took over the editorship in 1980 and served the journal well over the years. With his leadership, the number and quality of the papers increased to its present healthy state. I must also extend a big thank you to Conrad Mueller and the University of the Witwatersrand who pioneered desk top publishing of QI in August 1985, using the IBM mainframe and its laser writer. Without Conrad's diligence and the excellent facilities provided by the Wits Computer Centre and subsequently the Computer Science Department, QI would easily have degenerated into a second-rate magazine. Quintin Gee, also of the Wits Computer Science Department, has taken over from Conrad and has raised the production quality of QI to new heights, as this issue testifies.

I look forward to your help and support in the future. Long live QI!

Judy M Bishop
Editor
June 1988

A Detailed Look at Operating System Processes

B H Venter
Department of Computer Science, University of Fort Hare, Private Bag X1314, Alice, Republic of Ciskei

Abstract

An operating system provides, among other things, an operational definition of a process. The concept of a process is one of the fundamental concepts of Computer Science, and the designer of an operating system must strive to provide a definition that is simple to understand, does not violate the intuitive notions one has about processes, and is simple to implement efficiently on a wide range of computer systems. On the other hand, the definition should not fail to provide the functionality that existing operating systems have, by user demand, gradually evolved into providing. This paper presents a framework for discussing the operational definition of a process, and uses this framework to discuss systematically some of the more important decisions and trade-offs regarding processes, that the designer of a new operating system must make.
***Keywords**: operating system, process, light-weight process, memory-sharing, interrupts, exceptions, real-time*
***Computing Review Category**: D.4.1 OPERATING SYSTEMS, Process Management*

Received June 1987, Accepted July 1987

1. Introduction

There is no universally agreed upon definition for the concept of a process, and the definitions offered in the literature are often inadequate (for example, those in [1]). And while it is difficult to fault abstract definitions such as [2], they are not specific enough to be of much use to an operating system designer.

An operating system amounts to an operational definition of a process, and no two operating systems amount to exactly the same definition. The designer of a new operating system must strive to provide a definition that is simple to understand and use, does not violate one's intuitive notions about processes, and is simple to implement efficiently on a wide range of computer systems. On the other hand, the definition should not fail to provide the functionality that existing operating systems have, by user demand, evolved into providing.

The aim of this paper is to provide a framework for discussing operational definitions, particularly with regard to functionality, and then to use this framework to discuss systematically some of the more important decisions and trade-offs regarding processes, that a designer of a new operating system must make.

The considerations outlined in this paper have formed the basis for the process concept offered by a new operating system that the author has designed [5]. The process concept of this operating system differs significantly from the definition offered by UNIX [4,6], a system that many proponents proclaim as suitable for adoption as THE standard operating system, but which offers an inadequate definition of a process.

However, this paper is not a critique of UNIX or any other operating system, nor a defence of the author's proposal. The paper aims to identify and discuss the general principles involved, and readers are invited to make their own comparisons and to draw their own conclusions.

2. The Framework

The most basic property of an operating system process is that it executes on a virtual processor created by the operating system. Each such virtual processor has its own set of registers and has access to a subset of the memory of the real processor used to implement it. Furthermore, the instructions of a virtual processor can be regarded as being extended with extra instructions, namely the system calls offered by the operating system, which allow it to interact with the devices and other virtual processors in its environment. A process can thus be defined as a virtual processor seen in conjunction with the set of instructions (program) that it is executing.

The differences between the operational definitions offered by different operating systems can be described as differences in the capabilities of the virtual processors created by these systems.

In the following sections, the desired capabilities of a virtual processor are discussed in the light of a list of questions. Some of these questions may seem to have trivial answers, but all have been answered in different ways by different operating system designers. The list is:

* Should two different virtual processors be able to access the same area of physical memory?

* Should virtual processors use virtual or real addresses to access memory?
* Should virtual processors be able dynamically to vary the amount of memory allocated to them?
* Should a virtual processor be interruptible? That is, should a non-deterministic event in its environment be able to start the execution of an interrupt service routine?
* Should a virtual processor be able to handle exceptions caused by instructions executed by it?
* To what extent should a virtual processor be controllable by an external agent?
* What should the initial state of a newly created process be?
* Should it be possible to make assumptions about the rate of execution of a virtual processor in 'real time', as well as about the elapsed 'real time' needed to finish executing a routine after the occurrence of an event in its environment?

3. Memory Sharing

Data structures kept in shared memory areas have long been the principle concept used to achieve co-operation among different processes. However, along with the obvious advantages, allowing processes to share memory has certain disadvantages:

* Processes are implicitly given access to their private memories as part of their creation. However, to provide processes with access to shared memory areas, an operating system has to provide an explicit mechanism. Such a mechanism must provide a naming scheme and must not allow security to be compromised. Allowing memory sharing thus adds complexity to an operating system.

* Two processes cannot share a memory area efficiently unless their virtual processors can be implemented by physical processors with access to a common physical memory. This either limits the class of hardware that can be used, or necessitates an additional mechanism to control the physical processors used to implement virtual processors - thus adding more complexity to the operating system.

* Further mechanisms must be provided to allow processes to synchronise their access to shared memory areas. It is difficult, if not impossible, to make these mechanisms general, safe, and efficient at the same time. Moreover, they add yet more complexity to the operating system.

It is possible for an operating system designer to accept these disadvantages as unavoidable and to provide generalised mechanisms that allows arbitrary processes to share memory (as is done in UNIX System V [6]). However, since non-shared memory multi-processors (for example, hundreds of workstations inter-connected by means of a high speed local area network) are likely to become more common, if not prevalent, it makes sense to avoid memory sharing among processes. It therefore also makes sense to ask whether it is necessary to provide complex, generalised mechanisms to allow arbitrary processes to share memory.

Memory sharing only makes sense if the sharing processes are

a) executed by a single physical processor, or

b) executed by separate physical processors that share access to a common physical memory.

a) Assuming that the processes that are co-operating via memory sharing are all executed by the same physical processor, the question arises why they should not be consolidated into a single process. The multiple processes cannot together execute faster than a single process since the available real processor cycles remain fixed. In fact, the overhead incurred by extra context switches and process synchronisation make such multi-process formulations slower than single process formulations.

The main argument for having multiple co-operating processes on the same physical processor is that multiple-process solutions can sometimes be simpler to program than single-process solutions. A typical case is a server process that may receive requests from different clients in quick succession, each of which may involve waiting for some event to happen in the environment of the server (for example the completion of a disk access). Unless a single-process server explicitly breaks up each request into multiple subrequests, none of which require waiting, and interleaves the execution of different requests on this complicated basis, the single-process server will be slower than a multi-process server since context switching among the various co-operating processes automatically achieves the desired interleaving.

However, it is straightforward to incorporate a local scheduler into the code of a single-process server. This makes it possible to formulate the server as a set of memory-sharing, co-operating 'light-weight' processes, while actually executing the server as a single operating system process. Since the 'light-weight' scheduler is 'user code', it can dispense with many of the precautions that the operating system 'heavy-weight' scheduler must take. It can therefore be much more efficient to construct a set of memory-sharing, co-operating processes without the aid of expensive operating system mechansims.

There will, of course, be cases where memory-sharing, co-operating processes need the separate identities and relative isolation available only to operating system, 'heavy-weight' processes. However, these cases should be comparatively rare, and confined mainly to 'systems programming' situations. It thus seems reasonable to require these processes to be declared as 'privileged and trusted' (by suitably authorised users) and to be explicitly tied to specific physical processors. It is then possible to provide a simplemechanism that dispenses with most of the complications of generality and security checking to enable such privileged processes to obtain access to sharedmemory areas.

Another use for memory sharing among processes is to allow them to access common blocks of code, such as language-provided run-time systems. However, such code sharing need not affect the logic of a process, and the issue can thus be relegated to a convention between the linker and loader, rather than be treated as part of the definition of a process. In some systems, memory sharing is also used by debuggers. However, such debuggers can be accommodated by providing 'privileged and trusted' server processes that allow them to achieve the same ends.

b) Assuming now that there are multiple physical processors that have access to a shared physical memory, and therefore that the memory-sharing processes can be executed in true parallel, the 'light-weight' process solution is no longer applicable, and the 'privileged and trusted' solution is too primitive and restrictive. Furthermore, the existence of 'teams' of closely co-operating processes executing in true parallel opens up new opportunities and complications, such as 'co-scheduling' [3] and non-blocking ('spin-lock') forms of synchronisation.

It is, however, possible to accommodate such hardware without resorting to generalised mechanisms by extending the 'light-weight' process solution. On non-shared memory systems, the routines for implementing a 'light-weight' scheduler will be provided as a standard library, to be linked into the code of the 'heavy-weight' process hosting the collection of 'light-weight' processes. However, on shared-memory multi-processors these routines will simply invoke 'hidden' system calls, which see to it that the 'light-weight' processes are executed on different physical processors, while otherwise being part of a single 'heavy-weight' process.

The discussion on shared memory can be summarized as follows. It is easier and more efficient to implement a process concept that does not allow processes to share memory. Furthermore, an application formulated as a set of processes that do not share memory can be executed on a wider range of computer systems, which is highly desirable.

However, when it is essential to exploit a shared-memory multi-processor, this can be achieved by introducing 'light-weight' processes that are 'internal' to normal 'heavy-weight' processes. These 'light-weight' processes can also be supported on a single processor by incorporating an internal 'user-code' scheduler into the code of a single 'heavy-weight' process. Thus, applications that explicitly exploit shared-memory multi-processors can still be ported to other kinds of computer systems, albeit with a performance penalty in some cases.

It should be noted that the concept of a 'light-weight' process is really independent from the concept of a 'heavy-weight' process. The rest of this paper is exclusively concerned with 'heavy-weight' processes.

4. Virtual Address Spaces

A virtual address space typically allows a virtual processor to view its subset of the physical memory as one or more contiguous blocks of memory, starting at fixed addresses such as zero.

Providing virtual processors with such virtual address spaces has several disadvantages: it limits the class of suitable hardware, most processor caches must be invalidated after each context switch, and the operating system is complicated by the need to cope with different address spaces.

However, appropriate memory management hardware units are becoming common, and virtual address spaces facilitate the writing of compilers. Furthermore, virtual address spaces allow a process to comprise a number of disjoint areas of physical memory and make it possible to provide processes with virtual memories that are larger than the physical memory (however, this is becoming increasingly less important as physical memories become larger).

Moreover, widely-used operating systems such as UNIX provide processes with virtual address spaces. It may thus be more difficult to port programs developed for existing operating systems to a new operating system that does not provide processes with virtual address spaces.

Thus, the advantages of providing virtual address spaces appear to outweigh the disadvantages.

5. Dynamic Memory Allocation

Allowing a process to change the amount of memory available to it dynamically is highly desirable. For example, a process such as a compiler then needs to use no more memory than is required by the input of a particular run, and a process such as a sort utility can use as much (or as little) memory as is available during a particular run.

However, to be implemented efficiently, dynamic memory allocation requires memory management hardware that allows the virtual processor to access a potentially large number of disjoint segments of real memory. Furthermore, it complicates both the operating system resource allocation policies and the code of the dynamically sized processes, since the amount of memory needed by a process is not known in advance and it may be impossible to grant a request for extra memory.

These difficulties can be ameliorated effectively by requiring the linker to supply the loader (via parameters in the process image) with the maximum amount of extra memory a process may request, as well as the minimum amount of extra memory the operating system must guarantee. These parameters allow particular implementations of an operating system to overcome limitations of the memory management hardware by pre-allocating or partially pre-allocating extra memory for a process when it is created. They also facilitate the writing of dynamically sized processes as a minimum amount of memory can be guaranteed to be dynamically allocatable.

6. Virtual Interrupts

Whether or not to make virtual processors interruptible is a somewhat contentious issue. Anyone who has programmed an interruptible processor will recall at least one extremely hard-to-find error caused by subtle interactions between non-deterministic interrupts. None but the best and bravest (or most foolish) programmers will venture anything of substance that the final 'debugged' system running on an interruptible processor will be completely free of errors.

The main argument for having interrupts is that they allow fast responses to external events and avoid the need to check repeatedly whether the event has occurred. However, these properties can be obtained in a non-interruptible system by packaging interrupts as arriving messages, by providing a process with suitable mechanisms for suspending its execution pending the arrival of a message, and by providing mechanisms that enable a process to respond quickly to the receipt of a message.

The only processes for which these 'no interrupt' mechanisms are clearly less suitable than interrupt-based mechanisms are the device driver processes that must ultimately deal with the real interrupts on the real machine. These interrupts should preferably be dealt with as rapidly as the hardware allows, and the less software packaging is involved, the better. However, since device driver processes can reasonably be required to be designated as 'privileged and trusted', it suffices to give such processes access to the real interrupts via a simple (but dangerous) mechanism involving no overhead for other processes.

As will be seen below, the programmer who really wants an ordinary process to be asynchronously interruptible can readily achieve this by means of a protocol between the process and its parent. Furthermore, the entire mechanism and the overhead of the mechanism are firmly under the control of the programmer.

7. Exceptions

Real processors often use the same mechanism to handle exceptions and interrupts. However, since exceptions are usually caused by a processor trying to execute an erroneous instruction, it makes no sense for an operating system to package an exception as a message to the process, and then to allow the virtual processor to carry on executing until the process explicitly accepts the message.

Clearly, after an exception, the virtual processor must suspend the execution of the instruction stream that caused the problem, and either carry on with a different instruction stream, or have the process terminated. The mechanism for doing this should preferably involve minimal overhead for the ordinary process, which should not generate exceptions and should be terminated when it does. The mechanism should also be machine independent so as not to decrease portability, and should provide the programmer or language run-time system with complete control over what happens when an exception is generated.

The simplest mechanism that will do the job is as follows. When a process generates an exception, it is suspended, and the event is packaged into a message that is sent to its parent process. The parent of a process can then cause its suspended subprocess to be restarted at any point within the subprocess address space. Alternatively, the parent can simply terminate the subprocess.

This mechanism allows the operating system to handle exceptions using a simple non-interrupting scheme that exacts no overhead other than the cost of sending a message (the minimum cost in a non-shared memory multi-processor). The policy decisions on what to do about an exception are relegated to a machine-independent protocol between the process and its parent process. Moreover, the mechanism provides the programmer and/or language run-time system with complete control over what happens when a process generates an exception.

8. Control by External Agents

External agents, such as devices and users, can always be represented by corresponding processes. Thus the issue is the extent to which one process can control another. As previously indicated, a process can restart a suspended subprocess, as well as terminate a subprocess. It can also be informed whenever a subprocess terminates itself or is suspended.

It may seem unreasonable to restrict the privilege of controlling a process to its parent process. However, restricting the privilege avoids the need to introduce an explicit 'right to control' granting mechanism and makes it easier to check whether a process has the right to make a control request (especially in a distributed implementation of the operating system). Also, a programmer can synthesise control facilities in non-parent processes by letting the parent execute requests communicated to it by these processes. The restriction, therefore, simplifies the operating system without impairing functionality. Furthermore, the overhead of providing extended control rights is limited to those applications for which these are required.

An additional control feature often provided by operating systems is the ability to cause an interrupt or exception in another process. This can be achieved by allowing a process to suspend a subprocess, whereafter it can restart the subprocess at the address of the interrupt/exception handler. This allows, for example, a process to extricate a subprocess from an endless loop, and to restart it inside a 'cleanup and reset' procedure.

Allowing a process to be suspended at an arbitrary point in its execution, and then to be restarted at another, causes some complications. Firstly, the 'interrupt' handler entered when the suspended process is restarted may eventually wish to resume the interrupted control flow (in true interrupt handler fashion). Thus, when a process is suspended, the address of the instruction that was about to be executed must be saved on the process stack. Secondly, the interrupt handler may need to know the reason for the interrupt. Consequently, the parent must supply a reason code when it suspends a subprocess; this code is then stacked along with the 'return' address. (When a subprocess suspends itself, it too supplies a reason code, andwhen it is suspended because of an error, the system provides a reason code.)

Suspending a subprocess so that it can be resumed cleanly from the point of interruption can be very difficult if the subprocess is inside a system call at the time that the suspension request is received (which can be at any time on most multi-processors). Consequently, a suspension request issued by a parent while the subprocess is inside a system call takes effect only when the system call returns. If the parent proceeds to restart a subprocess that is still waiting to be suspended, the completion of the parent's restart request is delayed until the subprocess can be suspended.

However, if a subprocess is inside a system call that may never return, for example when it is waiting for an input/output operation that will never be completed, then such a scheme will cause the parent to be delayed indefinitely when it tries to extricate its subprocess from an indefinite delay. Consequently, such system calls must be interruptible.

When interrupted inside such calls, the stacked reason code is modified, and additional information is placed on the stack. An interrupt handler that wishes to return to the point of interruption can use this stacked information to resume the interrupted system call. On the other hand, an exception handler that must extricate the process from an erroneous control flow, rather than resume it, must be able to regard an interrupted system call as completed.

The complications that arise when a parent is allowed to suspend its subprocesses raise the question whether it is possible to dispense with this mechanism. However, the only alternative to suspending and restarting a run-away subprocess is to terminate it. When a process is terminated, all of its communication links are severed and all its subprocesses are terminated - otherwise, other complications would arise.

The cascading effects of terminating a process forming part of a set of co-operating processes may be too severe to be acceptable to a multi-process, real-time, fault-tolerant application. Albeit somewhat complicated, the semantics of suspend/restart seem preferable. And, even though complicated to describe, the suspend/restart mechanism can be implemented without exacting much overhead cost.

With regard to the automatic termination of subprocesses, note that a process can nevertheless create another process that will survive itself. A process would do this by not creating the new process as its own subprocess, but by requesting an operating system provided non-terminating 'server' process to create and control a subprocess on its behalf. Such a server would provide the functions of the background batch queues found in many operating systems.

9. The Initial State of a Process

There can be little argument that it is desirable to control fully the initial state of a process. However, there are at least two ways of achieving this. One is to allow the system linker to create a fully specified process image in a file, and for the loader to load the image, exactly as specified, into memory. Another way, taken by UNIX, is to duplicate the current image of a process when it creates a subprocess, and also to allow a process (usually a new subprocess) to replace its current image with one specified in a file.

However, creating a new process by duplicating the memory image of the parent process is not sensible if the subprocess immediately replaces the image with one from a file (by far the most common case). Furthermore, such an unnecessary duplication can be a very expensive operation if the image must be copied from one physical memory to another via a network.

Once a new process is created, it can either immediately start executing, as is the case for a UNIX process, or it can remain in a suspended state until explicitly started by its parent process. Such an initial suspension allows a parent process to create several subprocesses, and to set up communication links between them, while they are in a known state. A parent can also connect the communication ports of subprocesses to servers, files, or devices, or transfer the use of some of its own communication links to subprocesses. Initial suspension is thus preferable.

Another initialisation issue concerns the interface of a new process to its environment. A new UNIX process inherits copies of all the file descriptors of its parent. This is fine when the descriptors provide read-only access to actual files. However, if one of the file descriptors represents a terminal and both parent and child use it simultaneously, chaos results. In the operating system designed by the author, the UNIX concept of a file descriptor is superceded by a generalised 'communication link', and it can be both complicated and undesirable automatically to provide a new process with duplicates of all the communication links of its parent. Consequently, a new process is created without any communication links, and it is up to its parent explicitly to provide the links that the new process expects to be able to use without establishing them itself.

10. Rate of Execution

If the operating system is to support real-time applications, or even just interactive applications, there is little choice but to allow a programmer to make some assumptions about the rate of execution of a process. It is also necessary to allow the rate of execution to be influenced by the programmer.

Most operating systems allow priorities to be attached to processes, and some implement feed-back schemes that dynamically adjust the priorities of processes. While workable, and often successful, this approach does suffer from the drawback that priorities must be chosen and controlled carefully. When the number of real-time processes become large, it becomes very difficult to arrive at an appropriate set of priorities. And when processes are dynamically created and destroyed in response to the demands of interactive users, it becomes very difficult to use priorities with predictable effect.

A simpler approach is to allow the rate of execution of a process to be specified directly, rather than be influenced indirectly via priorities. For example, the speed of a processor can be rated on some absolute scale and the rate of execution of a process expressed as a number on the same scale, with the following interpretation. Let Y be the speed of the processor and X the desired rate of execution of a process. The process should then receive at least X/Y of the processor's cycles, say every 100 milliseconds, while the process remains ready to execute.

Of course, to guarantee that a process will progress at least as rapidly as at its desired rate of execution, the sum of the rates of all the processes that can be ready at the same time must not exceed the speed of the processor. However, it should be easier for a real-time application designer to ensure that this is true, than it would be to ensure that a set of priorities will result in the desired progress by each process.

To aid real-time application designers further, an operating system should also be able to distinguish among 'real-time' and 'ordinary' processes, with real-time processes pre-empting ordinary processes. A real-time application designer then need not take non-real-time processes into account at all.

11. Conclusion

In striving to provide the simplest possible definition of a process, while providing as much functionality as possible, the impact of various trade-offs must be evaluated carefully.

This paper has identified a number of the more important aspects of functionality and has discussed the issues and trade-offs involved, particularly in terms of the simplest mechanisms that can be used to provide the needed functionality.

By providing the necessary functionality from the start, it is possible to avoid the problems associated with piecemeal evolution. In this regard, it is instructive to compare the complex definition of a process offered by UNIX System V to the simple, but incomplete, definition offered by early versions of UNIX.

References

[1] H.M. Deitel, [1984], *An Introduction to Operating Systems, Addison-Wesley,* Reading Mass.
[2] J.J. Horning and B. Randell, [1973], Process Structuring, *ACM Computing Surveys,* **5** (1), 5-30.
[3] J.K. Ousterhout, [1981], *Medusa: a distributed operating system,* UMI Research Press, Michigan.
[4] D.M. Ritchie and K. Thompson, [1978], The UNIX time-sharing system, The Bell System Tech. Journal, **57** (6), 1905-1929.
[5] B.H. Venter, [1987], The design of a new general-purpose operating system, Ph.D. Thesis, University of Port Elizabeth.
[6] XELOS Programmer Reference Manual Perkin-Elmer Corporation, New Jersey, 1984.

This paper first appeared in the Proceedings of the 4th South African Computer Symposium.

A New General-Purpose Operating System

B H Venter
Department of Computer Science, University of Fort Hare, Private Bag X1314, Alice, Republic of Ciskei

Abstract

The current generation of widely-used, multi-user, general-purpose operating systems have evolved from versions that were designed when many of the issues that are important today were unimportant or not even thought of. This evolution has not been totally successful. In particular, the current generation is ill suited for implementation on loosely-coupled multi-processors. A new operating system, designed with current requirements in mind, and flexible enough to adapt successfully to likely future requirements, has been developed as part of a project to build a loosely-coupled multi-processor system that should have the performance and functionality of a 'super main-frame' computer. This paper concentrates on describing the fundamental mechanisms of the operating system: processes, inter-process communication, and servers. It also briefly outlines the support provided for data security, database applications, and real-time applications.
Keywords: *operating systems, system calls, inter-process communication, distributed systems, operating system security, operating system database support, real-time.*
Computing Review Category: *D.4 OPERATING SYSTEMS*

Received June 1987, Accepted July 1987

1. Introduction and Motivation

A modern general-purpose operating system can be expected to provide a high level of data security, to provide appropriate support for a comprehensive database management system, and to support real-time applications. Furthermore, such a system can be expected to isolate applications from the underlying hardware. That is, the operating system should be implementable on most current and future hardware systems, and an application written in a standard high level language should be able to run on any hardware system running under the control of such an operating system.

Moreover, a computer controlled by a modern operating system should be able to form part of a network of distributed computers, and provide users with efficient, transparent access to the resources available via the network. In particular, a modern operating system should be able to make effective use of the loosely-coupled multi-processor systems that are beginning to appear.

The current generation of widely-used, multi-user, general-purpose operating systems have gradually evolved from versions designed in the 1970's and 1960's. Then, many of the issues that are of considerable importance today, were unimportant or not even thought of. In particular, loosely-coupled multi-processor systems were unforeseen, and operating systems were designed with a single-processor mind set, making it difficult to port these systems to loosely-coupled multi-processors

For example, current implementations of UNIX cannot even be ported to tightly-coupled multi-processors without major changes and preferably a complete rewrite [2]. The situation is even worse for loosely-coupled multi-processors: the latest 'standard', UNIX System V [4], allows arbitrary processes to share memory which cannot be done efficiently on a loosely-coupled multi-processor. Furthermore, the message-passing mechanism which is of critical importance to distributed applications - is far too complicated and cumbersome to be implemented efficiently. In fact, the designers of this message-passing mechanism have apparently assumed that the sender and receiver processes share access to a common physical memory. There are also other aspects of UNIX which, upon close examination, prove difficult or impossible to implement effectively on a loosely-coupled multi-processor.

The situation is not much better when one considers other widely-used current generation operating systems. Thus, if one aims to build a new computer system based on a loosely-coupled multi-processor, a substantial operating system development effort must be undertaken.

If compatibility with existing software is one's principal concern, one should aim to implement either a UNIX 'look-alike', or an MVS 'look-alike'. However, as pointed out above, UNIX is not well suited to the role. The same is true for MVS.

If, on the other hand, making do with limited resources is one's principal concern, then it makes sense to develop a new operating system, with full use being made of what has been learnt about operating systems since the 1970's.

The author is currently involved in the development of a loosely-coupled multi-microprocessor system. The aim is that this system should provide 'super main-frame' functionality and performance. The project is being undertaken with relatively limited resources, and building a working system with the available resources is considered more important than providing compatibility with some existing software base. Consequently, a new operating system has been designed for this computer, and a fairly complete 'quick-and-dirty' prototype has already been implemented. A full-scale implementation, using the prototype operating system as the development system, is currently under way.

The rest of this paper is a brief survey of the main features of the operating system design. The intention is not rigorously to justify design decisions, nor to point out what contribution the work makes, but rather to provide the reader with an overall description and enough detail to compare the new operating system to any existing operating system. A more detailed description can be found in [3].

2. Processes

A new process can be created either by forking an existing process, in the style of UNIX, via the following two system calls: ('->' means 'returns')

fork_process (process_num, monitoring_io_port)
-> process_num replace_image (file_num, entry_point)

or by loading the process' starting image directly from a file:

load_new_process (file_num, monitoring_io_port)
-> process_num

The latter method is more appropriate for a loosely-coupled multi-processor. It makes little sense to copy the current memory image of the parent process over the network to the processor that is to execute the new subprocess, only to replace it soon afterwards with a new image obtained from a file, as is usually the case. Forking is only provided to facilitate UNIX compatibility.

Note that, unlike UNIX, a process can fork not only itself, but also one of its subprocesses. Furthermore, a parent process can be informed of all state changes in its subprocesses, and can exert complete control over them, with the ability to terminate, suspend, or restart a particular subprocess.

Furthermore, unlike UNIX, a new process is created in a suspended state, and execution must be started explicitly by its parent. This allows a process to load several subprocesses and to set up communication links between them while they are in a known state.

A parent process may also suspend an executing subprocess and then restart execution at a different instruction. The suspend/restart mechanism is formulated such that the restarted subprocess can execute either an interrupt handler (eventually returning to the point of interruption), or an exception handler (never returning).

As is to be expected, a process can suspend or terminate itself and supply a reason code that will be received by its parent. A process can also dynamically obtain additional memory from the operating system, and release unused memory for use by other processes.

The operating system does not normally allow processes to share access to the same area of memory, since, in general, it is not possible or desirable to ensure that processes execute on physical processors that have access to a common physical memory bank. Memory sharing is therefore discouraged by limiting it to specially privileged 'server' processes. (Server processes are discussed in Section 4.)

It is, however, possible to simulate a form of memory sharing between different 'light-weight' processes executing on the same processor by incorporating a scheduler into the code of a single 'heavy-weight' operating system process. Such 'light-weight' schedulers can ignore most of the fairness, synchronisation, and security issues that an operating system must address, and can thus provide a much cheaper form of single-processor concurrency than the operating system. A typical user of 'light-weight' processes would be a server process that must serve several clients concurrently.

3. Inter-Process Communication

As Hoare pointed out [1], transferring information from a process to a process does not differ from transferring information from a device to a process, or from a process to a device. In fact, in a modern operating system implementation, device drivers are likely to be implemented by processes.

Consequently, the operating system provides a generalised I/O call as the principal means whereby processes must interact with their environment. This system call corresponds to the classical I/O call of current generation operating systems in most respects, generalising it only in so far as to allow I/O operations to be performed on processes as well as on files and devices, and by allowing a single operation to perform both output and input.

The I/O call is invoked as follows:

io (io_port, operation, address, out_len, out_buf, in_len, in_buf)

Note that all the parameters, except out_buf, may be updated by the call.

An I/O port corresponds to a UNIX file descriptor, but may represent another process, as well as a file or device. Furthermore, up to sixteen different I/O operations may be carried out on an I/O port. For example, when an I/O port is linked to a file, the following operations are supported:

0 = read next string (length given in in_len)
1 = write next string (length given in out_len)
2 = read string at offset from file start (given in address)
3 = write string at offset from file start
4 = append string at end of file
5 = get current length of file (result in in_len)
6 = set current length of file to length in out_len
7 = flush updates to non-volatile storage

As can be seen, some operations are input operations, others perform output, and still others do neither. It is also possible to have operations that perform output as well as input. In general, the subset of operations that can be carried out on an I/O port and the effect of the operations depend on the kind of object to which the I/O port is linked. When an I/O port is used as an inter-process communication link, the programmer has full control over the subset of allowable operations and the interpretation given to them.

An I/O port is linked to a file, device, or server via:
open (process_num, io_port, object_num, desired_ops) -> available

Note that a process can open not only its own I/O ports, but also those of subprocesses; object_num identifies the file/device/server; desired_ops is a bit map indicating the subset of the sixteen possible operations that the caller wishes to carry out on the I/O port. (For example, to open a file for sequential read-only access, desired_ops must have a value of 0000000000000001.)
desired_ops) -> available
An I/O port is linked to a process via:
connect (process_a, io_port_a, bitmaps_a, time_out_a, process_b, io_port_b, bitmaps_b, time_out_b)

A process may connect the I/O ports of its subprocesses to each other, or may connect its own I/O ports to the I/O ports of subprocesses. Thus a process can communicate with its subprocesses, siblings, parent, and files/devices/servers.

Note that each I/O port has a bit map associated with it, indicating which I/O operations may be carried out on it, as well as a bit map indicating which I/O operations perform output. Additionally, a time-out is associated with each port. These values are explicitly specified for ports opened with connect, but are determined by the object to which the port is linked for ports opened with open.

The general progression of an I/O call is as follows:

- send operation,address,out_len,in_len to the other process (append contents of out_buf if the operation calls for it)
- wait for a response from the other process (return with an error code if no response within time_out)
- return operation,address,out_len,in_len sent by the other process, and store rest of its response in in_buf (sender out_len = receiver in_len, and sender in_len = receiver out_len)

When an I/O port has just been connected to another, the first operation that outputs the contents of out_buf is delayed until the other process performs an input operation on its corresponding I/O port. The I/O call of the process that performed the input operation then completes, returning the parameters and buffer contents supplied by the process that performed the output operation (resulting in a data transfer taking place; see also figure 1). Meanwhile, the outputting I/O call is suspended while awaiting a response. This response is received when the process that performed the initial input operation performs its next I/O operation. The response consists of the parameters and possibly the contents of out_buf, supplied to the second I/O operation.

Thus, two communicating processes are always synchronised so that the initiation of an operation by one, causes the completion of an operation by the other, as well as a data transfer. The result is that communication takes on a 'hand-shaked' request-response nature, and that when one process sends output to another, the other has already set up an input buffer to the receive the transferred data. This eliminates the need for a system buffer pool and takes care of flow-control and synchronisation.

When a port is opened to a file/device/server, the operating system engages in a dialogue with a corresponding file-driver-process/device-driver-process /server-process, which results in an inter-process communication link being set up between the client process and a driver/server process. After the open, the driver/server is waiting to complete an I/O operation, and the client process has yet to perform its first operation.

When the client performs its first I/O operation on the port, the incomplete I/O at the driver/server completes, resulting in the driver/server receiving the request made by the client. After serving the request, the driver/server responds by initiating a next I/O operation, which in turn causes the client's incomplete I/O operation to complete, returning the desired results.

Naturally, the I/O call allows a proceed option (indicated by setting a modifier bit in operation). A proceed I/O call sends the request to the other process, but does not wait to receive the response. The requesting process can later complete the call and. receive the response by performing another I/O call on the port – setting a modifier bit to indicate whether or not to wait for the response, if this has not yet been received.

It is possible for a process to have a number of I/O ports on which proceed I/O operations were carried out. Therefore, it is possible to perform an I/O call that will complete any one of these incomplete calls. This facility will typically be used by a server process that serves many clients concurrently, and thus may be waiting for several requests (that is, have several incomplete I/O calls) at the same time.

The I/O call also supports broadcasting/multi-casting, as well as scatter/gather transfers.

4. Servers

Server processes are the principal means by which the operating system provides its services. Furthermore, servers can be used as the basic blocks for building distributed applications.

A server process has a globally visible name, drawn from the same name space used for files and devices. Thus, a prospective client establishes an inter-process communication link with a server by calling open. The operating system carries out a call to open by sending an unsolicited message to the target of the open call. Whether it is a file, device, or explicit server, the target of an open call is always a process; hence the term 'server' will from now on be understood to include files and devices.

When a server starts executing, it sets up a number of 'reconfigurable' I/O ports, using:

make_reconfigurable (io_port, valid_op_bitmap, in_len, in_buf, time_out)

While reconfigurable, an I/O port does not represent a communication link to any particular process, but acts as a receiving port for unsolicited messages. Thus, when the operating system sends an unsolicited message to a server, one of the server's reconfigurable I/O ports is selected to hold the message, and the server will receive the message when it tries to complete an I/O call on that port (it will usually try to complete an I/O call on any port). Note that make_reconfigurable sets up the buffer to hold the unsolicited message.

The unsolicited message received by the server

a) can be trusted because it comes from the operating system
b) fully identifies the prospective client, its privileges, the type of access required, and so on.

After receiving such a message, the server must decide whether or not to accept the client and indicate this by outputting an appropriate message through the I/O port that received the request. If the client is accepted, the I/O call indicating the acceptance remains incomplete until the client performs its first I/O operation on the port it has just opened successfully. Thus, when a client is accepted via a reconfigurable I/O port, the I/O port is configured as a dedicated inter-process communication link between client and server. Note that, among other things, the server's acceptance message supplies information to be associated with the client's I/O port: the valid operations, the operations that output out_buf, and the time-out to be used when waiting for a response from the server.

A server process can be tied to a particular processor of a multi-processor system, in which case it is loaded when that processor bootstraps. Alternatively, a server can be 'untied', in which case it is loaded into any available processor when first referenced by an open call. Untied servers usually terminate when they have no more clients.

Designating a process as a server, thus giving it global visibility, is a privileged operation since a server is trusted to take part in the 'new client' protocol. Servers are also the only class of processes that can be allowed to perform certain privileged operations.

For instance, a server that is tied to a particular processor can be used as a device driver by allowing it the privilege to access I/O space and to install interrupt handlers. A tied server can also be allowed to share memory with other servers tied to the same processor.

It follows that an implementation of the operating system will itself largely consist of a set of server processes distributed among the various physical processors. Adding new servers to the collection making up the basic operating system will be straightforward, resulting in an 'open', extensible, adaptable operating system. Furthermore, structuring the operating system as a set of servers communicating via messages greatly facilitates the transparent integration of local resources (services) with resources available, via a network, from other systems.

The server mechanism also allows a single service (that is, a single object in the name space) to be implemented by several co-operating processes. One way to implement such a multi-process server is to designate several server processes as the members of a single server group, identified by a single 'group object' number. When a client performs an I/O operation on an I/O port linked to a group, the request message is broadcast to all members of the group. It is possible for the structure of the group to be invisible to the client, in which case the members must use a protocol to ensure that only one response will be generated. Alternatively, a client may be required to be aware of the group structure, in which case the client must set up additional I/O ports to receive the multiple responses (an I/O port can receive at most one response for each request).

It is also possible to implement a server group, without using broadcasting, by means of a co-ordinator process. In this case, client requests go to the co-ordinator, and the co-ordinator then 'subcontracts' them to the other processes co-operating to provide the service. A co-ordinator can subcontract all client interaction, in which case the 'request to become a client' message goes to the co-ordinator and is subcontracted, following which all further interaction is between client and subcontractor (unbeknown to the client). Alternatively, a co-ordinator can subcontract individual requests, inwhich case all requests first go to the co-ordinator and then to the subcontractors.

Note that subcontracting and broadcasting can be combined. It is thus possible to exploit multiple processors to achieve both speed and fault-tolerance, while providing clients with the illusion of dealing with a simple 'single' server.

5. The File System

The file system is intended to facilitate the storage and retrieval of arbitrary strings of bytes, such as text files, object files, and process images. The file system is not intended as an efficient or convenient store for records since it is more reasonable to use a database management system to store and retrieve such data.

The file system is implemented as a set of communicating servers, with each open file having a corresponding 'driver' process that actually receives and acts upon the I/O operations that client processes perform on I/O ports linked 'directly' to files. The file system servers are structured as a hierarchy and in such a way that the database management system can exist 'alongside' the file system, by making use of the lower level 'disk block' servers instead of the file system visible at system call level.

Files are identified by numbers drawn from a global name space that includes devices and servers. This allows user interfaces to use arbitrary symbolic name-to-file number mappings, further adding to the 'open', extensible, adaptable nature of the operating system.

Files are allocated in two steps:
create_temporary_file (logical_vol_num, size_hint)
-> file_num make_file_permanent (file_num,
future_expansion-hint)

Temporary files are automatically reclaimed when the process that allocated them terminates. By converting these temporary files into permanent files, rather than directly allocating permanent files, it is possible to follow a protocol that results in the allocation of a file and the recording of a symbolic name-to-file number mapping as an atomic operation.

The size hints that may be given when a file is allocated allow the operating system to pre-allocate space for a file so that sequential access is optimised. The file system need not heed the hints and will never refuse to allocate or grow a file because a suitable run of disk blocks is not available.

The design of the file system attempts to minimise the visibility of physical disk volumes to application programmers in order to promote program transportability: All permanently on-line storage media are consolidated into logical volume zero and files from all volumes are identified from a single name space.

However, it is necessary to introduce the concept of distinct volumes to cope with removable disk packs. Clearly, the operating system cannot just incorportate removable disk packs into a global pool and allow arbitrary files to be allocated on removable disk packs. Consequently, removable disk packs are grouped into one or more logical volumes, any of which can be off-line. File allocations on these volumes must be indicated explicitly and can only be performed by suitably authorised users.

6. Security

Users gain access to the operating system by interacting with a user authentication server. It is possible to associate an arbitrary authentication server with a given user access device, and to restrict a given server to admitting only a subset of users. Thus, it is possible to make it reasonably easy to gain entry as a casual user, while making it arbitrarily difficult to gain entry as a privileged user.

When a user gains entry into the system, the corresponding user access device is assigned to an appropriate user interface process, which is 'executing on behalf of' the user. The 'executing on behalf of' property of a process is inherited by all subprocesses, and can only be changed if a process is a specially privileged server process (for example, an authentication server). Thus, all actions initiated by a user are carried out by processes authentically executing on behalf of the user.

Files, devices, and servers all have owners, and are accessible only to processes executing on behalf of the owner or a user explicitly mentioned in an access list - a file listing all the users that may access the protected object, with each entry indicating an individual set of privileges for the user. (Note that a server may reject a client even if the client is listed in its access list.)

Additionally, files, devices, servers, and processes are associated with 'information categories'. An arbitrary number of information categories can be established, and the operating system enforces a set of rules that ensure that information cannot 'flow' from one category to another, unless specifically permitted. This is basically achieved by limiting a process to accessing only objects associated with the same information category, while also allowing a process to have read-only access to objects associated with categories that have flow paths to the category of the process. A process may only change its category if its user is permitted to operate in the new category and if the change cannot result in an illegal flow of information.

The operating system also includes mechanisms for limiting software piracy, denial of resources, Trojan Horse attacks, and the use of covert channels to subvert information flow controls.

7. Database Support

Database managers are principally supported by the server mechanism. Furthermore, the file system is structured in such a way that a database manager can access disk blocks directly.

The only additional support provided by the operating system is in the form of a transaction co-ordinating server. This server is accessed by client processes via the system calls:

```
start_transaction
commit_transaction -> success_indicator
abandon_transaction
```

These calls keep the transaction co-ordinator up to date on the transaction status of processes, and it, in turn, keeps 'transaction supporting' servers up to date on the transaction status of their clients.

The transaction co-ordinator co-ordinates a two-phase commit, provides a 'centralised' deadlock detection service, and allows transaction supporting servers to use a wide range of synchronisation strategies, including optimistic strategies.

While it is logically a centralised service, the transaction co-ordinator can be implemented as a distributed set of co-operating processes, and thus does not preclude the implementation of a distributed database manager.

8. Real-Time Support

The operating system allows the minimum rate of execution of a process to be specified directly rather than be influenced indirectly by means of priorities. Furthermore, processes are classed as 'real-time' or 'ordinary', with real-time processes pre-empting ordinary processes. All operating system server processes execute as ordinary processes, thus giving real-time application programmers complete control over processor allocation.

Note also that server processes can be granted the privilege of handling interrupts directly, as well as to share memory with similar servers on the same processor. Furthermore, the operating system can be configured with a separate 'real-time' file system that serves only real-time processes and thus isolates real-time processes from interference by non-real-time processes in this aspect as well.

9. Conclusion

This paper has briefly outlined the design of an operating system that is hardware independent, provides a high level of data security, and supports distributed applications, database applications, as well as real-time applications.

This has not been achieved by introducing radically new concepts, but by carefully reformulating, generalising, and integrating the basic concepts that any operating system must support.

Firstly, the process concept has been stripped of restrictions and assumptions, leaving a straightforward mechanism that can be utilised efficiently for a wide range of applications. Secondly, the input/output mechanism has been generalised to provide a straightforward, efficient form of inter-process communication. Thirdly, the concept of a device has been generalised into the concept of a server, on which the operating system itself is largely based. Servers make the operating system flexible and extendible, and provide a suitable mechanism for distributed implementations.

References

[1] C.A.R. Hoare, [1978], Communicating Sequential Processes, *Communications of the ACM*, **21** (8), 666-677.

[2] M.D. Jannsens, J.K. Annot and A.J. Van de Goor, [1986], Adapting UNIX for a multiprocessor environment, *Communications of the ACM*, **29** (9), 895-901.

[3] B.H. Venter, [1987], The Design of a new general-purpose operating system, Ph.D. Thesis, University of Port Elizabeth.

[4] XELOS Programmer Reference Manual, Perkin-Elmer Corporation, New Jersey, 1984.

This paper first appeared in the Proceedings of the 4th South African Computer Symposium.

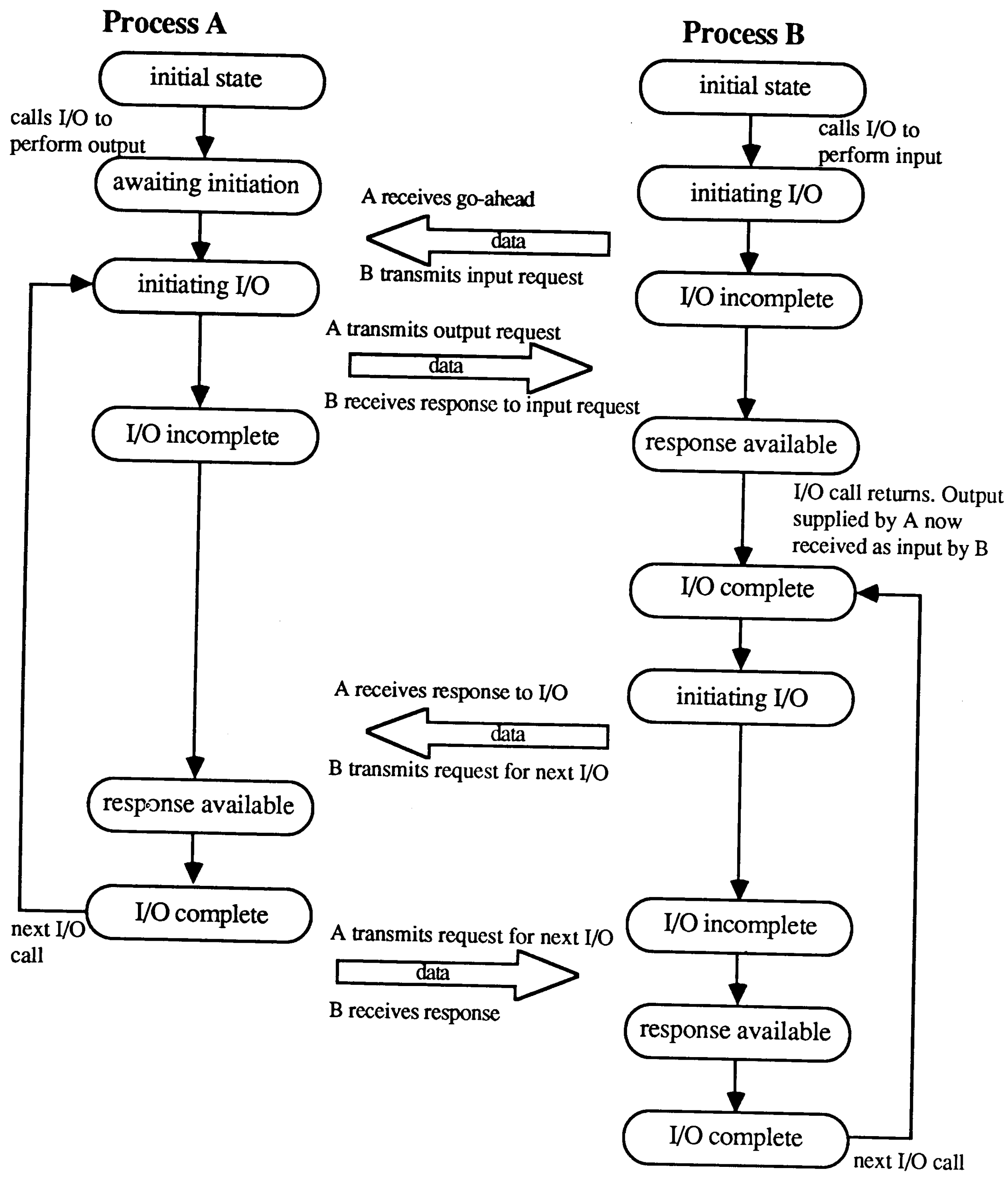


Figure 1 I/O state transition diagram

Protection Graph Rewriting Grammars and the Take/Grant Security Model

S H von Solms and D P de Villiers
Department of Computer Science, Rand Afrikaans University, PO Box 524, Johannesburg, 2000

Abstract

The operations in the Take/Grant Protection Model are formalised using theory and results from the discipline of formal languages. A Protection Graph Rewriting Grammar is defined, which generates protection graphs consistent with the restrictions inherent in the Take/Grant Model.
Keywords: *Take/Grant Model, protection graph, formal grammar, rewriting system*

Received July 1987, Accepted July 1987

1. Introduction

The goal of this paper is to define a graph rewriting grammar to simulate the operations in the Take/Grant security model [2]. This grammar will have certain context conditions applicable to every production in the grammar. These context conditions will allow the grammar to simulate the conditions inherent in the different operations allowed in the Take/Grant model.

The grammar will rewrite one protection graph [1] by another, the rewriting process being controlled by the context conditions of the productions.

For a discussion of the Take/Grant model, the reader is referred to [1], [2].

2. Protection Graphs

2.1 Definition

A protection graph is a directed, loop-free, edge labelled, two colour graph

$P = (V,R)$

where $P = V \cup O$, $S \cap O = \varnothing$, is called the set of nodes, with S the set of subject and O the set of object nodes.

Edges are labelled by nonempty subsets of a finite set of labels

$R = \{r_1, ..., r_n\} \cup \{t,g\}$, called *rights.*

R contains two distinguished elements t and g. We say P is a protection graph over V/R.

2.2 Definition

For any protection graph $P = (V,R)$, let $\psi(V/R) =$

$\{\underset{x}{\bullet} \xrightarrow{R_1} \underset{y}{\bullet} \mid x,y \;\varepsilon\; V, R_1 \subseteq R\}$,

i.e. $\psi(V/R)$ is the set of all protection graphs over V/R consisting of only two nodes.

An element $D \;\varepsilon\; \psi(V/R)$ is called a *limited* protection graph.

x is called the initiator and y the receiver of the limited protection graph

$D = \underset{x}{\bullet} \xrightarrow{R_1} \underset{y}{\bullet}$ denoted by i(D) and r(D) respectively.

3. Protection Graph Rewriting Grammars

3.1 Definition

A protection graph rewriting grammar (PGR-grammar), is a system

$G = (V,\Sigma,P,I,R)$ where

V is a finite non-empty set of nodes, $V = S \cup O$

S is called the subjects, and O is called the objects.

Σ is a finite non-empty subset of V, called the set of terminal nodes.

P is a set of productions described below.

I is the initial protection graph (axiom).

R is a finite non-empty set of edge labels, with the two distinguished labels t and g in R.

P consists of 4 types of productions:

3.1.1 Growth production

A growth production can rewrite (replace) a node by a limited protection graph, i.e. it can extend (let grow) an existing protection graph. (Add a new node to the protection graph). Growth productions have the following form:

$(x, D, (X_1 ; X_2), (Y_1, Y_2))$

where $x \;\varepsilon\; V$, D is a limited protection graph over V/R with $x = i(D)$

$X_1, X_2 \subseteq \Sigma$

Y_1, Y_2 are limited protection graphs over V/R

$r(D) \;\varepsilon\; X_2$.

A production of this kind is applied in the following way to a protection graph H containing x:

Suppose $D = \underset{x}{\bullet} \xrightarrow{R_1} \underset{y}{\bullet}$, $R_1 \subseteq R$.

Add y as a node to H, with a new directed edge, labelled R_1 between x and y, if the following conditions hold:

(a) All elements of X_1 appear somewhere in H,
(b) No elements of X_1 appear anywhere in H,
(c) All elements of Y_1 appear as subgraphs somewhere in H,
(d) No elements of Y_2 appear as subgraphs anywhere in H.

X_1 and X_2 are known as the permitting/ forbidding node contexts respectively, and Y_1 and Y_2 as the permitting/ forbidding edge contexts.

From the description above, it is clear that this type of production allows the protection graph under consideration to grow, i.e. add new nodes.

Note y ε Y_2 (from the definition of this kind of production). This requirement prevents the addition of y if y already appears in the relevant graph.

3.1.2 Edge generation productions

An edge generation production can generate (insert) an edge between two existing nodes in a protection graph.

Edge generation productions are of the form:
$(x, y, R_1, (X_1, X_2), (Y_1, Y_2))$, where
$(x, y \,\varepsilon\, V, R_1 \subseteq R, X_1, X_2, Y_1, Y_2)$ are defined and used as in 3.1.1.

The effect of such a production is to insert the edge labelled by R_1 between nodes x and y if the context conditions are satisfied.

Note that x, y ε X_1, i.e. both x and y must appear in the protection graph under consideration. Further we demand that

$\underset{x}{\bullet}\xrightarrow{R_i}\underset{y}{\bullet}$, $\forall R_i \subseteq R$ be in Y_2, i.e. there may not (already) exist an edge between x and y.

3.1.3 Edge removal productions

An edge removal production can remove an existing edge between two nodes.

These productions have the form:
$(x, y\ (X_1, X_2), (Y_1, Y_2))$
with x, y, X_1, X_2, Y_1, Y_2 defined as in 3.1.1.

The effect of such a production is to remove the edge between nodes x and y if the context conditions are satisfied.

Note that $\underset{x}{\bullet}\xrightarrow{R_1}\underset{y}{\bullet}$, must be in Y_1 i.e. there must be an existing edge, labelled R_1, between x and y.

3.1.4 Edge label update productions

These productions can change the label of an existing edge.

Their form are:
$(x, y, R_1, R_2, (X_1, X_2), (Y_1, Y_2))$,
where x, y, X_1, X_2, Y_1, Y_2 are defined as in 3.1.1, and $R_1, R_2 \subseteq R$.

The effect of such a production is to change the label of the edge between x and y from R_1 to R_2.

Note that $\underset{x}{\bullet}\xrightarrow{R_1}\underset{y}{\bullet}$, must be in Y_1 i.e. there must be an existing edge, labelled R_1, between x and y.

3.2 Definition (Informal)

The language generated by a protection graph rewriting grammar

G = (V, Σ, P, I, R)

is the set of all protection graphs over Σ/R that can be generated from the axiom I using productions from P.

3.3 Definition

If Σ = V, we call G an extended protection graph rewriting grammar.

In the rest of this paper we will assume, without stating it every time, that Σ = V.

We will also not distinguish explicitly between object and subject nodes.

4. Note

We have now generated different protection graphs from the axiom. This generation was strictly controlled by the context conditions of the different productions.

The "definition" of a PGR-grammar is based on [3].

5. Simulating the Take/Grant Model with PGR-Grammars

In this paragraph we will take the four rewriting rules in the Take/Grant model [2], and simulate them with PGR-productions. We will discuss each rewriting rule separately, primarily to show how the context conditions in the PGR-production "controls" the specific Take/Grant rewriting rule.

5.1 The Take-rule

Let x, y and z be nodes in a protection graph PG_1, such that x is a subject. Let there be an edge from x to y, labelled R_1, such that "t" ε R_1, and an edge from y to z labelled R_2.

Let $R_3 \subseteq R_2$.

Then the "take"-rule defines a new protection graph PG_2 by adding an edge to PG_1 from x to z.

Graphically

$$\begin{tikzcd} x \arrow[r, "R_1"'] & y \arrow[r, "R_2"'] & z & \Rightarrow & x \arrow[r, "R_1"'] \arrow[rr, bend left, "R_3"] & y \arrow[r, "R_2"'] & z \end{tikzcd}$$

We can simulate this using an edge generation production as discussed in 3.1.2. Consider the following edge generation production:

5.1.1 (x, z, R_3, (y ;),

$$(\underset{x}{\bullet} \xrightarrow{R_1=\{t\}\cup\bar{R}_1} \underset{y}{\bullet}\ ,\ \underset{y}{\bullet} \xrightarrow{R_2} \underset{z}{\bullet}\ ;\ \underset{x}{\bullet} \xrightarrow{R_4} \underset{z}{\bullet}))$$

$\forall$ ε S, (remember V = S ∪ 0), $R_3 \subseteq R_2$, $\forall R_4 \subseteq R$.

Production 5.1.1 states:
The node y must appear in PG_1
The limited protection graphs

$$(\underset{x}{\bullet} \xrightarrow{R_1=\{t\}\cup\bar{R}_1} \underset{y}{\bullet} \text{ and } \underset{y}{\bullet} \xrightarrow{R_2} \underset{z}{\bullet}$$

must appear in PG_1, and the limited protection graph

$$\underset{x}{\bullet} \xrightarrow{R_4} \underset{z}{\bullet}$$ may not appear for any $R_4 \subseteq R$, i.e.

there may be no existing edge between x and z in PG_1. So PG_2 is generated from PG_1 using production 5.1.1.

Note that 5.1.1 is actually a shorthand notation for a whole set of productions. Every member of the set can be written down explicitly, making the semantic requirements viz
x ε S, $R_3 \subseteq R_2$, $\forall R_4 \subseteq R$ unnecessary.

Using this set of productions we can therefore mechanically implement the Take-rule by some automata.

5.2 The Grant-rule
Let x, y and z be distinct nodes in protection graph PG_1 such that x is a subject. Let there be an edge from x to y labelled R_1, such that "g" ε R_1, and an edge from x to z labelled R_2. Let $R_3 \subseteq R_2$. The "grant"-rule defines a new protection graph PG_2 by adding an edge from y to z labelled R_3. Graphically

$$\begin{tikzcd} x \arrow[r, "R_1=\{g\}\cup\bar{R}_1"'] \arrow[rr, bend left, "R_2"] & y & z \arrow[r, phantom, "\Rightarrow"] & x \arrow[r, "R_1"'] \arrow[rr, bend left, "R_2"] & y \arrow[r, "R_3"'] & z \end{tikzcd}$$

Again we can simulate this using an edge generation production. Consider the following edge generation production:

5.2.1 (y, z, R_3, (x ;),

$$(\underset{x}{\bullet} \xrightarrow{R_2} \underset{z}{\bullet}\ ,\ \underset{x}{\bullet} \xrightarrow{R_1=\{g\}\cup\bar{R}_1} \underset{y}{\bullet}\ ;\ \underset{y}{\bullet} \xrightarrow{R_4} \underset{z}{\bullet}))$$

for x ε S, $R_3 \subseteq R_2$, $\forall R_4 \subseteq R$.

The explanation of this production follows the same lines as 5.1.1.

5.3 The Create-rule
Let x be any subject node in a protection graph PG_1, and let R_1 be a non-empty subset of R. The "create"-rule defines a new protection graph PG_2 by adding a new node n to PG_1 with an edge from x to n labelled R_1. Graphically

$$\underset{x}{\bullet} \Rightarrow \underset{x}{\bullet} \xrightarrow{R_1} \underset{n}{\bullet}$$

We simulate this using a growth production as described in 3.1.1. Consider the following growth production:

$$(x, \underset{x}{\bullet} \xrightarrow{R_1} \underset{n}{\bullet}; (; \{n\}) (;)$$

for x ε S, $R_1 \subseteq R$.

The forbidding node context checks that n does not already appear in PG_1, i.e. two identical nodes cannot appear in PG_2.

5.4 The Remove-rule
Let x and y be distinct nodes in a protection graph PG_1 such that x is a subject. Let there be an edge from x to y labelled R_1, and let R_2 be any subset of rights. The "remove"-rule defines a new protection graph PG_2 by deleting the R_2 rights from R_1.

Graphically,

$$\underset{x}{\bullet} \xrightarrow{R_1} \underset{y}{\bullet} \Rightarrow \underset{x}{\bullet} \xrightarrow{R_1 - R_2} \underset{y}{\bullet}$$

We can simulate this using an edge label update production as described in 3.1.4. Consider the following production: (Let $R_3 = R_1 - R_2$).

$$(x, y, R_1, R_3\ (;)\ (\underset{x}{\bullet} \xrightarrow{R_1} \underset{y}{\bullet};)$$

The permitting edge context requires that an edge between x and y, labelled R_1, must exist in PG_1.

If $R_3 = \varnothing$, the edge can be physically removed using an edge removal production as discussed in 3.1.3.

6. Conclusion and Further Research

From the approach taken in this paper, it seems possible to maintain a secure environment by using the PGR-productions to enforce the restrictions and context conditions inherent in any security policy. The decision making process, i.e. deciding which rights may be given to whom, and who may access whom, can be "hard-coded" into the system using the productions. The PGR-grammar can then automatically decide, by checking the situation of the present protection graph, and by applying applicable productions, where and when access is allowed.

The next step is to try to model the Send/Receive security model using the same principles.

References

[1] J. Biskup, [1984], Some Variants of the Take-Grant Protection Model, *Information Processing Letters*, **19**, 151-156.

[2] L. Snyder, [1981], Formal Models of Capability-Based Protection Systems, *IEEE Trans on Computers* **C-30** (3), 172-181.

[3] S.H. von Solms, [1984], Node-Label-Controlled Graph Grammars with Context Conditions, *International Journal of Computer Mathematics*, **13**.

This paper first appeared in the Proceedings of the 4th South African Computer Symposium.

Protocol Performance Using Image Protocols

P S Kritzinger
Department of Computer Science, University of Cape Town, Private Bag, Rondebosch, 7700

Abstract

Performance analyses of data communication systems do not always rely on a detailed analysis of the underlying protocols. Those analyses which do, usually rely on an analysis of the protocol state transition graph. These graphs tend to become very large for nontrivial protocols and the analyses correspondingly complex. Image protocols is a recent approach to reduce the complexity of communication protocol analysis. The method allows for the construction of an image protocol which is generally smaller than the original protocol and its analysis therefore less complex. An image protocol system is said to be faithful if it preserves the safety and liveness properties of the original protocol system. In this paper we show that an image protocol is also faithful as far as its performance is concerned, a result which simplifies performance studies of the protocol considerably. An example which illustrates the principles involved is included.
Keywords: *Protocol performance, image protocol, aggregation, state transition graph, finite state machine, multiclass queueing theory, queueing network, MVA-algorithm.*

Received July 1987, Accepted August 1987

1. Introduction

Most international standards for data communication networks are based upon a layered architecture. At each layer tasks or protocol entities provide and receive services from the adjacent layers and execute a peer to peer protocol between that layer and the corresponding layer at remote stations. These architectures are very complex and initial reports about their performance in terms of measures such as Data Unit throughput are not encouraging.

Performance analyses of data communication networks do not always include a detailed analysis of the protocol itself as it influences the execution of the protocol function. Those analyses which do, make use of a formal specification of the protocol and more specifically, the protocol state transition graph. As is the case for protocol validation methods, these graphs become very large for real-life protocols and the analyses correspondingly complex.

Despite the obvious advantages to be gained, protocol performance analysis is not a very active research area. The first work to make use of a formal specification is that of Bauerfeld [2] followed by Rudin [8]. Analyses based upon Petri-net descriptions are those by Molloy [6] and Razouk [9]. More recently the author has proposed a performance analysis technique [5] which applies the techniques of multiclass queuing network theory. The latter analysis is based upon the state transition graph of a protocol and although it can handle systems with many thousands of states the method can benefit from state reduction as indeed can all of the proposed methods.

Several techniques have been proposed in the literature for the reduction of the protocol state space with the objective of protocol validation in mind. Recent advances in this regard is the work on decomposition by Choi and Miller [3], that on phase reduction techniques by Chow et al. [4] and the use of image protocols by Lam and Shankar [10].

In this paper we show how the performance analysis technique proposed by the author [5] can be combined with the method of projections as proposed by Lam and Shankar to reduce the complexity of the performance analysis. We define the concept of faithful performance of an image protocol and show that it applies.

After first introducing the protocol system model of Lam and Shankar the performance analysis technique is described in terms of that model. A very brief overview is given of the construction of an image protocol and the paper ends with an analysis of the performance properties of an image protocol.

2. Protocol System Model

The reader is referred to Lam and Shankar [10] for a full description of the protocol system model used here. In this paper only those definitions from that paper which are required for an understanding of the performance model, described in the next section, are given.

Let there be I protocol entities $P_1, P_2, \ldots, P_I$ and K channels $C_1, C_2, \ldots, C_K$. Let S_i be the set of states of P_i and M_{ik} be the set of messages that P_i can send into C_k.

The dynamics of these protocol entities and channels are described by entity events and channel events.

Channel events specified for channel C_k are denoted by E_k and are used to model various types of channel errors. The reader is referred to [10] for a full description of channel event errors. The occurrence of a channel event in E_k depends on, and changes only the state of C_k; no change in the state of a protocol entity is involved and for that reason channel events are not considered any further in this paper.

There are three types of entity events:

- *Send Events:* Let $t_i(r,s,-m)$ denote the event of P_i sending message m into channel C_k where $m \in M_{ik}$ and C_k is in the outgoing channel set of P_i. The send event is enabled when P_i is in state r. After the event occurrence, P_i is in state s and m has been appended to the end of the message sequence in C_k.
- *Receive Events:* Let $t_i(r,s,+m)$ denote the event of P_i receiving message m from channel C_k where $m \in M_{ik}$ and C_k is in the incoming channel set of P_i. The receive event is enabled when P_i is in state r and m is the first message in channel C_k. After the event occurrence, P_i is in state s and m is deleted from the channel.
- *Internal Events:* Let $t_i(r,s,\alpha)$ denote an internal event of P_i where α is a special symbol indicating the absence of a message. This internal event is enabled when P_i is in state r. After the event occurrence, P_i is in state s. Internal events model timeout occurrences internal to P_i, as well as interactions between the entity and its local user.

Each send or receive event may alter the states of the entity and the channel involved in the event. Internal entity events do not affect the state of any channel. Following Lam and Shankar we use T_i to denote the set of events specified for P_i, $i = 1, 2, ..., I$.

In most cases, the behaviour of P_i would be nondeterministic. For example, we may specify T_i to contain both $t_i(r,s,x)$ and $t_i(r,u,x)$ where $s \neq u$ as well as both $t_i(r,s,x)$ and $t_i(r,s,y)$ where $x \neq y$. We denote the probability of an entity event $t_i(r,s,x)$ by $p_i(r,s,x)$. Note that unless otherwise specified, we will use the notation $t_i(r,s,x)$ to mean all of $x = +m$, $-m$ or α.

Each event corresponds to a set of transitions in the global state space G of the protocol system. The union of the sets of transitions over all events specified for the protocol system will be denoted by τ. The simultaneous occurrences of multiple events in the protocol system are treated as occurrences of the same events in some arbitrary order. The pair (G,τ) is also called the global state transition graph.

Figure 1 taken from [10] is an example of two protocol entities P_1 and P_2 modelled by two communicating finite-state machines.

3. Protocol Performance Model

Each event in the protocol system is specified by its enabling condition and its execution. The enabling condition of an event is a predicate in the components of the global state of the protocol system. We will not concern ourselves with the logical properties of a protocol system in this paper.

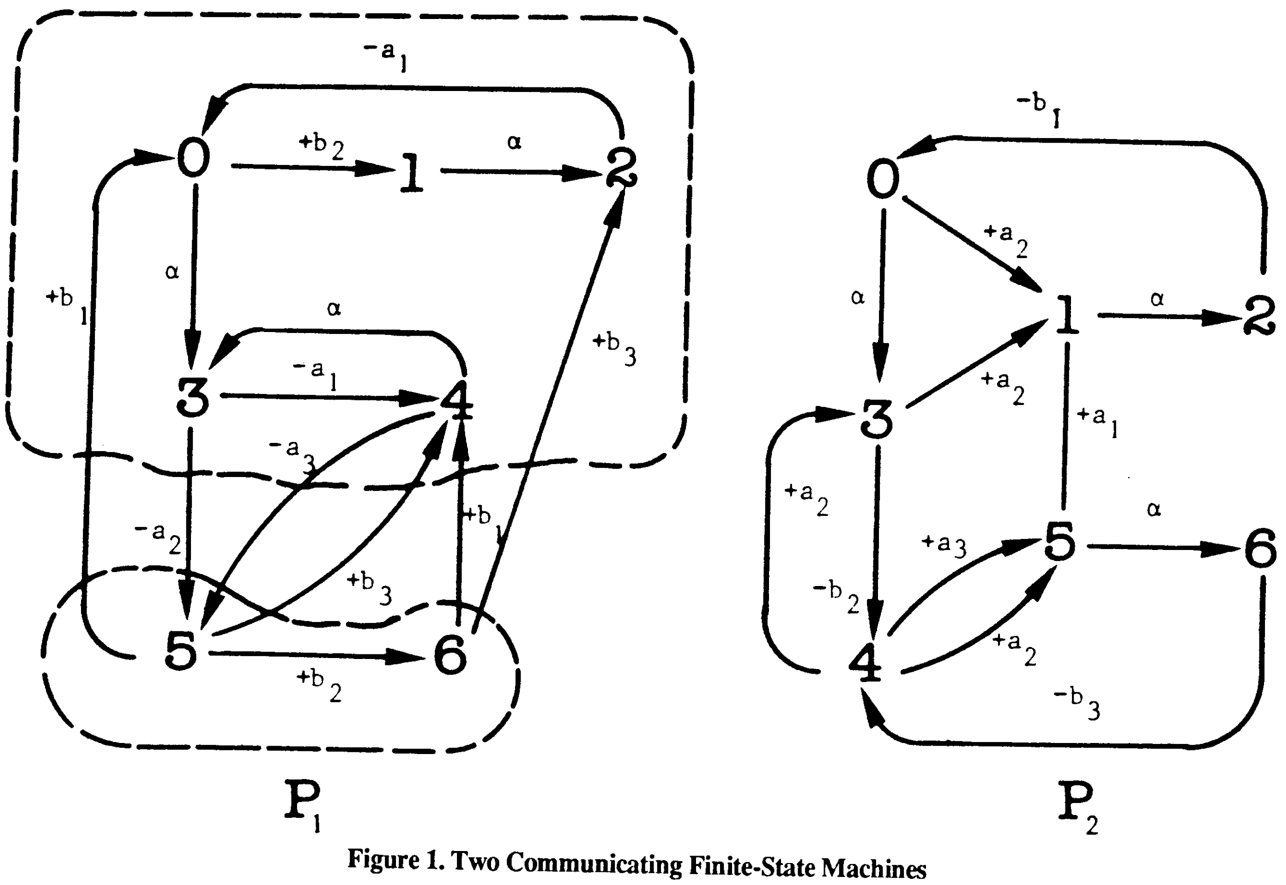


Figure 1. Two Communicating Finite-State Machines

The execution of an event however, specifies an update to the components of the global state and requires time from the processor on which the entity is executing.

The performance model first described in [5], assumes that there are l_i instances of entity P_i, $i = 1, 2, \ldots, I$ as would be the case with several endpoint connections in a real protocol system. It is moreover assumed that all protocol entities contend for service at a single processor. A measure of the performance of the protocol would typically be the rate at which a particular send event $t_i(r,s,-m)$ is executed or the throughput rate at the processor of all send events associated with entity P_i. These, and other performance quantities are computed by considering the protocol system as a closed multiclass queueing system with two service centres.

The one service centre is of the Processor Sharing type [1] and models the processor. The second service centre is a pure delay server and models the delays experienced in channels $C_1, C_2, \ldots, C_K$ associated with receive events.

Customers in the network are instances of the entities $P_1, P_2, \ldots, P_I$. A customer takes on a distinct class associated with each distinct entity event causing a state transition. Each distinct entity event therefore represents a distinct customer class. Each entity event is moreover classified to be either an *active event*, if the event requires processor time, or a *delay event* otherwise.

For example, all send events $t_i(r,s,-m)$ are active events, since they require the execution of a certain average number of instructions, which in turn requires a certain average amount of processor time. Receive events $t_i(r,s,+m)$ represent time delays in the corresponding channel. An internal event $t_i(r,s,x)$ could be either a delay event (denoted by $x = +\alpha$), representing a timeout event say, or an active event (denoted by $x = -\alpha$) when it represents an interaction between the entity and its local user requiring processor time.

Note that an event cannot be both an active event and a delay event; any such event would have to be decomposed into two events with a new intermediate state. If $t_i(r,s,\alpha)$ were such an event for example, it would be decomposed into an active event $t_i(r,s_1,-\alpha)$ and a delay event $t_i(s_1,s,+\alpha)$.

It follows from the above that we associate a time duration with every entity event. Denote the expected value of this time for event $t_i(r,s,x)$ by $\mu_i^{-1}(r,s,x)$.

In the model, a sequence of events will thus cause an entity to spend time in either one of the two servers, depending on whether the associated event is an active event or a delay event. For example, an entity in a transition due to active event $t_i(r,s,-m)$ having received a corresponding average service time $\mu_i^{-1}(r,s,-m)$ from the processor centre, will leave and either return to that same centre if the following event is again an active event, or would proceed to the delay centre if the following event is a delay event. In this way an entity would pass nondeterministically between the servers, depending on the sequence of events.

Figure 2 illustrates these concepts for the communicating finite state machine P_1 of Figure 1 where, arbitrarily, the internal events $t_i(r,s,\alpha)$ have been considered to be active events for all r, s and α.

Note that the only assumptions required to solve the model as a closed multiclass queueing network are that,

- the probabilities $p_i(r,s,x)$ are independent for all r, s and x, and
- the time $\mu_i^{-1}(r,s,x)$ associated with an event is independent of the time associated with any other event in the same protocol system.

4. Analysis of the Performance Model

As one event follows another, an entity P_i changes from one to the next, not necessarily different state in S_i. Let $\xi_i(r,,)$ denote the expected value of the number of times P_i is in state $r \in S_i$ between successive times that it is in some arbitrarily chosen state $r^* \in S_i$. In other words, $\xi_i(r,,)/\xi_i(r^*,,)$ is the relative number of times that P_i would be in state r compared to the number of times it would be in state r^*. The $\xi_i(r,,)$ satisfy the set of linear equations given by

$$\xi_i(r,,) = \sum_{s \in S_i} \sum_{x} \xi_i(s,,)p_i(s,r,x) \qquad (1)$$

where Σ_x denotes the summation over all $x \in M_{ik}$, $k = 1, 2, \ldots, K$, such that $t_i(s,r,x)$ takes entity P_i from state s to state r. It is not difficult to see that if $\xi_i(r,s,x)$ denotes the expected value of the relative number of times entity event $t_i(r,s,x)$ is executed, that

$$\xi_i(r,s,x) = p_i(r,s,x)\xi_i(r,,) \qquad (2)$$

Define the relative utilisation $v_{ia}(r,s,x)$ of the processor centre by instances of entity P_i in a transition due to active event $t_i(r,s,x)$ to be

$$v_{ia}(r,s,x) = \xi_i(r,s,x)/\mu_i(r,s,x) \qquad (3)$$

where $x = -m$ or $-\alpha$. Similarly we define the relative utilisation $v_{id}(r,s,y)$ of the delay centre by instances of entity P_i in a transition due to delay event $t_i(r,s,y)$ to be

$$v_{id}(r,s,y) = \xi_i(r,s,y)/\mu_i(r,s,y) \qquad (4)$$

where $y = +m$ or $+\alpha$.

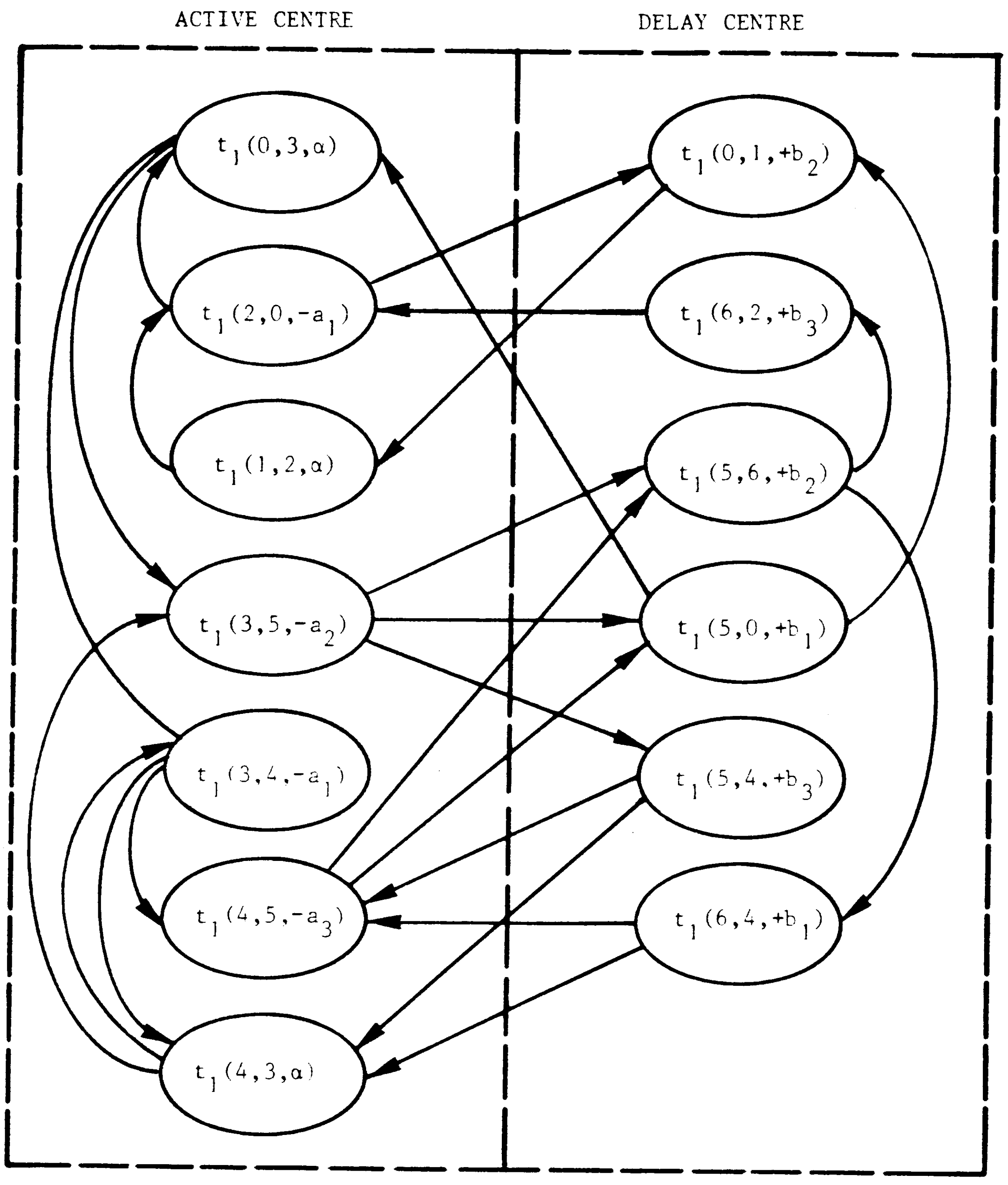


Figure 2. Event Sequences in the Two Servers for Entity P_1 of Fig. 1

Denote the total relative utilisation of the processor centre by P_i by

$$\rho_{ia} = \sum_{r,s \in S_i} \sum_{x} v_{ia}(r,s,x) \qquad (5)$$

where $x = -m$ or $-\alpha$, and of the delay centre by

$$\rho_{id} = \sum_{r,s \in S_i} \sum_{y} v_{id}(r,s,y) \qquad (6)$$

where $y = +m$ or $+\alpha$.

Let $\mathbf{L} = (l_1, l_2, \ldots, l_I)$ be the system population vector of the number of instances l_i of entity P_i, $i = 1, 2, \ldots, I$. Let $\chi_{ia}(r,s,x,\mathbf{L})$ denote the average rate at which active events $t_i(r,s,x)$ are completed at the processor or in other words, the expected throughput rate of the l_i instances of the entity P_i in a transition due to that event. It is known [11] that the value $\chi_{ia}(r,s,x,\mathbf{L})$ is given by

$$\chi_{ia}(r,s,x,\mathbf{L}) = \xi_i(r,s,x)\chi_i^*(\mathbf{L}) \qquad (7)$$

where $x = -m$ or $-\alpha$. The average throughput rate $\chi_{ia}(\mathbf{L})$ of entity P_i due to any active event $t_i(r,s,x)$ where $r,s \in S_i$, $x = -m$ or $-\alpha$, is the sum of the throughputs given by (7) for individual events. That is,

$$\chi_{ia}(\mathbf{L}) = \Xi_{ia}\,\chi_i^*(\mathbf{L}) \qquad (8)$$

with

$$\Xi_{ia} = \sum_{r,s\in S_i} \sum_{x} \xi_i(r,s,x) \qquad (9)$$

$\chi_i^*(\mathrm{L})$ in (7) and (8) is a mathematical quantity without direct physical interpretation and which is calculated in the Mean-Value-Analysis (MVA)-Algorithm [7] presented in Fig. 3.

```
Algorithm
Step 1. initialisation

  set $Q_a(0, 0, \ldots, 0) = 0$

Step 2. loop over the number of instances of every entity

  let $i = (i_1, i_2, \ldots, i_I)$

  for $i_1 = 0, 1, \ldots, l_1$ $i_2$; $= 0, 1, \ldots, l_2; \ldots; i_I = 0, 1, \ldots, l_I$ do;

($i_1$ changes most rapidly)

Step 3. loop for every entity
  for $j = 1, 2, \ldots, I$ do;
    $R_{ja}(i) = \frac{\rho_{ja}}{\Xi_{ja}}[Q_a(i - \mathbf{e}_j) + 1]$
    $R_{jd}(i) = \frac{\rho_{jd}}{\Xi_{jd}}$
    $\chi_j^*(i) = i_j/(\rho_{ja}[Q_a(i - \mathbf{e}_j) + 1] + \rho_{jd})$
  end;
  end of step 3.
    $Q_a(i) = \sum_{j=1}^{I} \Xi_{ja}\chi_j^*(i)R_{ja}(i)$
  end;
  end of Step 2.
```

Figure 3. The MVA-Algorithm

The average queue length $Q_{ia}(r,s,x,\mathrm{L})$ of instances of entity P_i in transition due to event $t_i(r,s,x)$, $x = -m$ or $-\alpha$ is given by

$$Q_{ia}(r,s,x,\mathrm{L}) = \frac{\xi_{ia}(r,s,x)}{\rho_{ia}} \Xi_{ia}\chi_i^*(\mathrm{L})R_{ia}(\mathrm{L}) \qquad (10)$$

The quantity $R_{ia}(\mathrm{L})$ in the last equation is the average response time at the processor centre of instances of entity P_i in any transition. According to Little's formula the corresponding quantity $R_{ia}(r,s,x,\mathrm{L})$ for entity P_i in transition due to an active event $t_i(r,s,x)$, $x = -m$ or $-\alpha$, is given by

$$R_{ia}(r,s,x,\mathrm{L}) = Q_{ia}(r,s,x,\mathrm{L}) / \chi_{ia}(r,s,x,\mathrm{L}) \qquad (11)$$

The same statistical quantities can be computed for the delay centre by replacing the subscript "a" by the subscript "d" in equations (7) - (11) and computing the relevant quantities for delay events.

The vector quantity $\mathbf{e}_j$ is a unit vector in the j-th direction. The reader is referred to [5] for a full description of the computation of these quantities using the MVA-algorithm.

5. Constructing an Image Protocol System

An image protocol system is a partition of each of the sets

S_i, M_{ik}, T_i, E_k for all i and k

in the original protocol system specification. Protocol entity states, messages and events in each partition subset are treated as equivalent and are aggregated to form a single quantity, called their image, in the image protocol system. Since partition subsets are mutually exclusive and collectively exhaustive, quantities that are treated as equivalent have the same image; quantities not treated as equivalent have different images. Following Lam and Shankar we shall use x' to denote the image of a protocol quantity x.

Since an image protocol is obtained from the original protocol by aggregations, it captures only part of the logical behaviour of the original protocol system. First, global states of the image protocol system correspond to aggregations of global states of the original protocol system. Second, in the global state space of the image protocol system, the observable effect of different events in the original protocol system may be different or nil. Our objective however, is to show that the performance behaviour of the image protocol system will be faithful to that of the original protocol system. Before doing that however, and for the convenience of the reader, we describe the process used in [10] to construct an image protocol system.

5.1 Aggregation of Entity States

An image state s' of a set of states of the original state space S_i is constructed by partitioning S_i. All entity states in a partition subset are aggregated to the same image. Let S'_i denote the set of images of states in S_i i.e., $S'_i = \{s' : s \in S_i\}$. S'_i is called the image state space of P_i.

For example, Fig. 1 shows the partition $\{\{0, 1, 2, 3, 4\}, \{5, 6\}\}$ of the state space S_1 of entity P_1. Let image state $0'$ denote the image of states 0, 1, 2, 3 and 4 in S_1 and $5'$ denote the image of states 5 and 6 in S_1. The image state space of P_1 is $S'_1 = \{0', 5'\}$. Similarly let image states $0'$, $1'$ and $2'$ respectively denote the images of the states in the partitions $\{0, 3, 4\}$, $\{1, 5\}$ and $\{2, 6\}$ of S_2. In this case $S'_2 = \{0', 1', 2'\}$.

5.2 Aggregation of Messages

Only when the reception of two messages m and n cause identical state changes in the image state space of the receiver of C_k are they treated as equivalent. This equivalence relation partitions M_{ik}, and messages within the partition subset may be aggregated to form an image message m'. The image message sets are given by

$M'_{ik} = \{m' : m \in M_{ik}\}$ for all i and k.

In particular, messages in M_{ik} whose receptions do not cause any state change in the image state space of the receiver are said to have a null image.

The reader is referred to the example in [10] for the derivation of the image message sets $M' = \{a_2', a_3'\}$, and $M'_2 = \{b_1'\}$ where b_1' denotes the image of messages b_1 and b_3 and the image of a_1 and b_2 are both null. These transitions in the image state spaces are illustrated in Fig. 4.

5.3 Aggregation of Entity Events

Entity events which have the same observable effect in the image global state space are aggregated to the same image entity event. An image event whose occurrence does not have any observable effect in the image global state space is said to be a *null image event*. An image internal event $t_i(s',r',\alpha)$ is a null image event if $s' = r'$. Image send events involving null image messages and infinite buffer channels are treated as image internal events in T'_i; in this case the image event is represented as $t_i(s',r',\alpha)$ and it is a null image event if $s' = r'$. Finally, image receive events involving null image messages must have $s' = r'$ by definition; hence such receive events are null image events if the channel involved is an infinite buffer channel.

The set of image events for P_i is defined by $T'_i = \{t_i(s',r',x') : t_i(s,r,x) \in T_i\}$ where $x = +m, -m$ or α and the image of $t_i(s,r,x)$ is not a null image event.

The reader is referred to the example in [10] for the derivation of

$$T'_1 = \{t_1(0',5',-a'_2), \{t_1(0',5',-a'_3), t_1(5',0',+b'_1)\}$$

and

$$T'_2 = \{t_2(0',0',+a'_2), t_2(0',5',-a'_3), t_2(0',1',+a'_3), t_2(1',2',\alpha), t_2(2',0',-b'_1)\}$$

in Fig. 5 which illustrates the image protocol constructed from the original protocol illustrated in Fig. 1.

6. Properties of Image Protocols

Lam and Shankar [10] discuss various logical properties of image protocols in relation to the logical properties of the original protocol.

In [5] the question was posed whether, depending typically on the constituent functions of the protocol, it would be possible to decompose a protocol state transition graph into its constituent

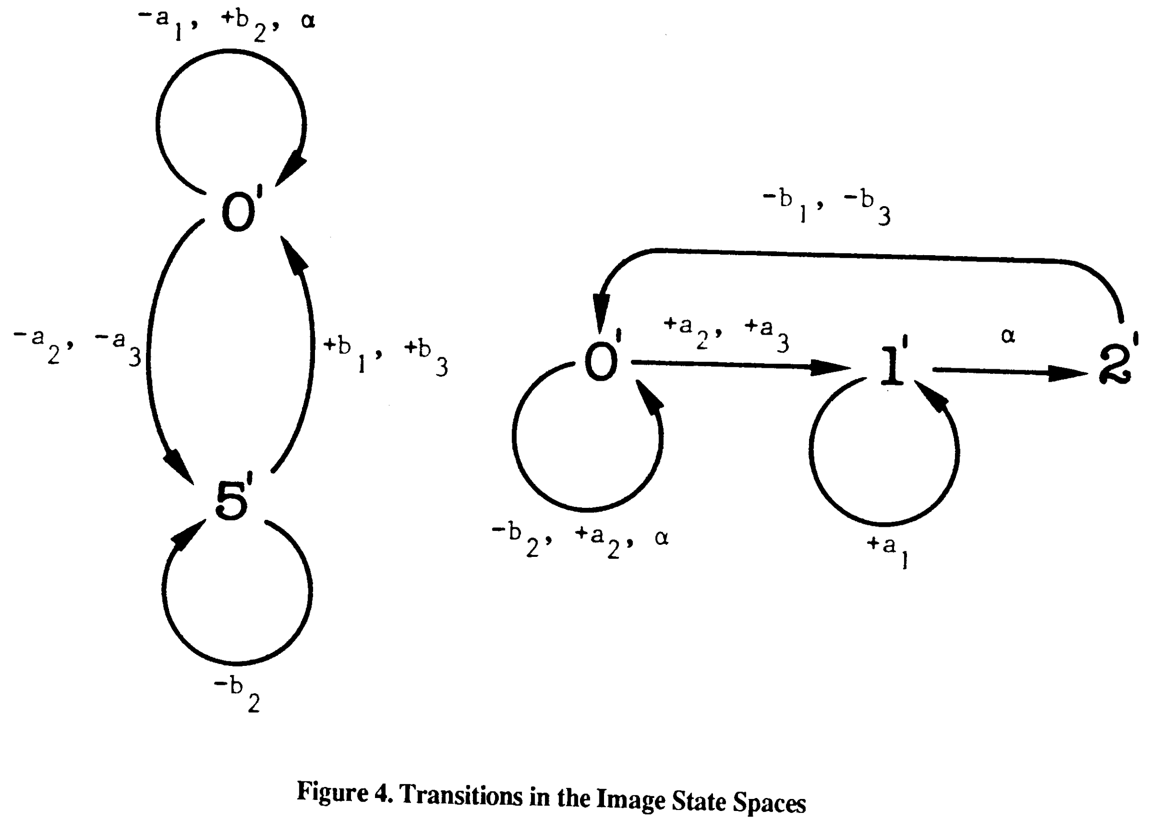


Figure 4. Transitions in the Image State Spaces

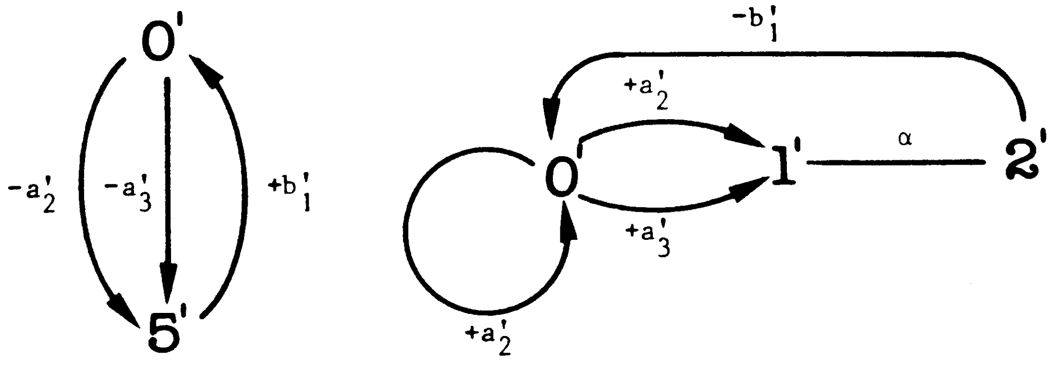


Figure 5. Image Protocol Constructed from the Original Protocol in Fig. 1

subgraphs, aggregate each subgraph and still have a system whose performance would be faithful to the original system. In this section we show that this is indeed possible.

6.1 Performance Properties

Let $t_i(r',s',x')$ denote the image entity event of a subset of entity send events in T_i. Define the relative utilisation of the processor centre by all instances of entity P_i in a transition due to event $t_i(r',s',x')$ to be

$$\nu_{ia}(r',s',x') = \sum_{(r',s',x')} \xi_i(r,s,x)/\mu_i(r,s,x) \qquad (12)$$

Here $\Sigma_{(r',s',x')}$ denotes the summation over all send events $t_i(r,s,x)$ in the original protocol system with image $t_i(r',s',x')$.

Similarly, let $t_i(r',s',y')$ denote the image entity event of a subset of entity receive events T_i. Define the relative utilisation of the delay centre by all instances of entity P_i in a transition due to event $t_i(r',s',y')$ to be

$$\nu_{id}(r',s',y') = \sum_{(r',s',y')} \xi_i(r,s,y)/\mu_i(r,s,y) \qquad (13)$$

where $\Sigma_{(r',s',y')}$ denotes the summation over all receive events $t_i(r,s,y)$ in the original protocol system with image $t_i(r',s',y')$.

Also let $\xi_i(r',s',x')$ denote the relative frequency of execution of the image event $t_i(r',s',x')$. Clearly

$$\xi_i(r',s',x') = \sum_{(r',s',x')} \xi_i(r,s,x) \qquad (14)$$

where $\Sigma_{(r',s',x')}$ has the same meaning as above. The quantities $\nu_{ia}(r',s',x')$, $\nu_{id}(r',s',y')$, and $\xi_i(r',s',x')$, are respectively the image protocol quantities corresponding to $\nu_{ia}(r,s,x)$, $\nu_{id}(r,s,y)$, and $\xi_i(r,s,x)$ of the original protocol defined in (2), (3) and (4).

Although all entity events $t_i(r,s,x)$ whose image is a null image event are eliminated in the construction of the protocol image system, they cannot be ignored in the analysis of the image protocol system. The effect of these events have to be taken into account in the computation of the performance quantities ρ_{ia} and ρ_{id} defined in (5) and (6). Let $T_i^{n-} \subseteq T_i$ denote the set of send events in P_i which have null images. Define

$$\rho_{ia}^{n-} = \sum_{T_i^{n-}} \nu_{ia}(r,s,x) \qquad (15)$$

where $x = -m$ or $-\alpha$. Similarly, let $T_i^{n+} \subseteq T_i$ denote the set of receive events in P_i which have null images. Define

$$\rho_{ia}^{n+} = \sum_{T_i^{n+}} \nu_{id}(r,s,x) \qquad (16)$$

where $x = +m$ or $+\alpha$.

Rewrite (5) and (6) in terms of image entity events,

$$\rho'_{ia} = \rho_{ia}^{n-} + \sum_{r',s' \in S'_i} \sum_{x'} \nu_{ia}(r',s',x') \qquad (17)$$

where $x' = -m'$ or $-\alpha$, and

$$\rho'_{ia} = \rho_{ia}^{n+} + \sum_{r',s' \in S'_i} \sum_{x'} \nu_{id}(r',s',y') \qquad (18)$$

where $y' = +m'$ or $+\alpha$, and

Definition: An image protocol system is said to have faithful performance if the value of a performance statistic of the entity P_i in a transition due to an image event $t_i(r',s',x')$ is the sum of the performance statistics of the entity events whose image is $t_i(r',s',x')$.

Note that if an image event is the image of a single event in the original protocol, that the statistics for the two events will be identical.

Theorem: The performance of an image protocol system constructed as explained above, is faithful.

Proof: The proof follows from the MVA-algorithm using the quantities ρ'_{ia} and ρ'_{id} defined in (17) and (18) instead of ρ_{ia} and ρ_{id} as well as the equivalent

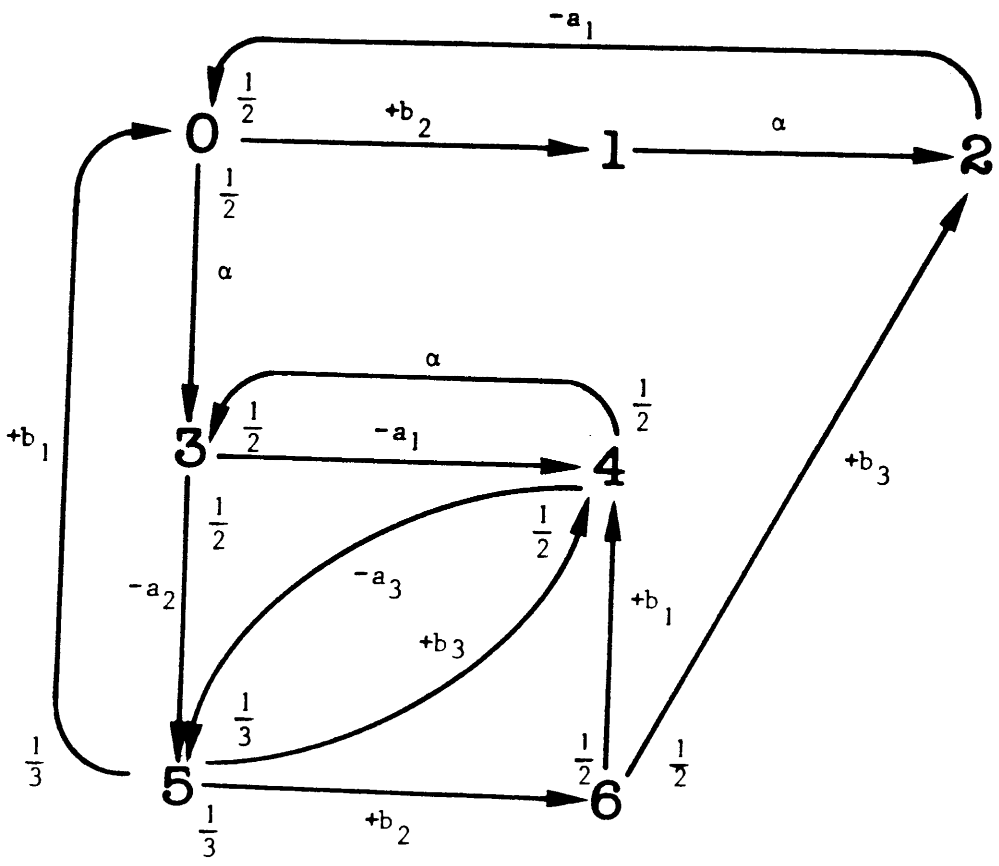


Figure 6. Finite State Machine with the Probability Values Indicated.

image protocol quantities defined in (12), (13) and (14) and substituting in the appropriate equations.

As an example, consider the throughput rate $\chi_{1d}(5',0',+b_1')$ at the delay centre of entity P_1 in transition due to image entity event $t_i(5',0',+b_1')$. Assume a single instance of P_1, the expected value of all times to be unity and the probabilities of the various events as illustrated in Fig. 6.

Computing the quantities given by equations (12) - (18) for the image protocol and applying the MVA-algorithm, it can be shown that $\chi_{1d}(5',0',+b_1')$ = 0.18182 per unit time from the image protocol equivalent equation (7). However, $t_1(5',0',+b_1')$ is the image of entity events $t_1(5,0,+b_1)$, $t_1(5,4,+b_3)$, $t_1(6,4,+b_1)$ and $t_1(6,2,+b_3)$. The equivalent throughputs are $\chi_{1d}(5,0,+b_1)$ = 0.06061 and $\chi_{1d}(5,4,+b_3)$ = 0.06061 and $\chi_{1d}(6,4,+b_1)$ = 0.03030 and $\chi_{1d}(6,2,+b_3)$ = 0.03030 per unit time. The result follows.

7. Conclusion

Previous work by other authors have shown how to construct image protocols for individual protocol functions of a complex protocol. The complexity of the image protocol system is less, and under certain conditions it is faithful to the original protocol system in all its safety and liveness properties.

Image protocols have a further advantage in that they reduce the complexity of analysing the performance of the protocol. In this paper it is shown that the performance of an image protocol system is also faithful to that of the original protocol system. It is therefore possible to do a performance study of the reduced image protocol system, where the results obtained would be the same as when the analysis were applied to the original protocol system.

References

1. F. Baskett, K.M. Chandy, R.R. Muntz and F.A. Palacios, [1975], Open, closed and mixed networks of queues with different classes of customers, *J. ACM*, 22 (2), 248-260.
2. W.L. Bauerfeld, [1983], Protocol performance prediction, *Proc Intnl Conf on Comm*, IEEE, Boston, 1311-1315, June.
3. T.Y. Choi and R.E. Miller, [1983], A decomposition method for the analysis and design of finite state protocols, *ACM SIGCOMM '83 Symposium on Communication Architectures and Protocols*, 167-176, October.
4. C.H. Chow, M.G. Gouda and S.S. Lam, [1985], A discipline for constructing multiphase communication protocols, *ACM Trans. on Computer Syst.*, 3 (4), 315-343, November.

5. P.S. Kritzinger, [1986], A performance model of the OSI communications architecture, *IEEE Trans. Commun.*, **COM-43**, 554-563.
6. M.K. Molloy, [1982], Performance analysis using stochastic Petri nets, *IEEE Trans. Comp.*, **C-31** (9), 913-917.
7. M. Reiser and S.S. Lavenberg, [1980], Mean-value analysis of closed multichain queueing networks, *J. ACM*, **27**, 313-322.
8. H. Rudin, [1984], An improved algorithm for estimating protocol performance, *Proc. IVth Workshop on Protocol Specification, Testing, and Verification*, Pennsylvania, June.
9. R.R. Razouk and C.V. Phelps, [1984], Performance analysis using timed Petri nets, *Proc. IVth Workshop on Protocol Specification, Testing, and Verification*, Pennsylvania, June.
10. S.S. Lam and A.U. Shankar, [1984], Protocol verification via Projections, *IEEE Trans. Software Engineering*, **SE-10** (4), 325-342, July.
11. S.S. Lavenberg, Ed., [1983], ch. 3.5 in *Computer Performance Modeling Handbook.*, New York: Academic Press Inc.

This paper first appeared in the Proceedings of the 4th South African Computer Symposium.

A Structural Model of Information Systems Theory

J Mende
Department of Accounting University of the Witwatersrand, PO Box 1176 Johannesburg 2000

Abstract

The laws and techniques of mature subjects such as Physics and Electrical Engineering are logically connected to one another in the form of a deductive network There is a foundation of basic laws deriving from them are successive layers of logically consequent laws and techniques The field of study "Information Systems' (I S) shares a common feature with Physics and Electrical Engineering All three provide the knowledge necessary for designing Man s artefacts Consequently we can expect their logical structure to be similar That means I S techniques should be logically connected to I S laws and I S laws should be logically connected to laws of fundamental subjects such as Computer Science Psychology and Management
***Keywords** information systems metatheory research*
***Computing Review Category** H O*

Received December 1987, Accepted April 1988

Islands of Knowledge

Since their invention some forty years ago, electronic computers have spread rapidly into many spheres of human activity Coupled with rapid advances in the technology itself, the enormous rate of computerisation has deluged the Information Systems researcher with urgent problems of practical systems development and management As a result, the academic field of study "Information Systems" (I S) has evolved rapidly, but not altogether soundly

In particular, many I S textbooks present their subject matter as independent "islands of knowledge" - much like the "islands of mechanisation" [1] prevalent in the business firms of the nineteen-sixties For instance, Gane & Sarson's "Structured Systems Analysis" [5] starts with several law-like statements in chapter 1 Yet there is no reference to those laws in the subsequent chapters on the techniques of data-flow diagramming, process-logic analysis, etc A similar omission occurs in Jackson's "Principles of Program Design" [6] The book describes a 'basic design technique' of data structure analysis, program structure formation and task allocation Yet this technique is almost entirely unsupported Jackson makes no attempt at logical derivation from underlying laws, indeed he does not even state such laws

Further examples of logical omission can also be found in Martin's "Strategic Data Planning Methodologies" [9], Lundberg's "Information Systems Development" [8], Weinberg's "Structured Analysis" [13] and G B Davis's "Strategies for information requirements determination" [2]

In order to identify and avoid such shortcomings, researchers should consider what kinds of knowledge the I S practitioner needs, and how different types of knowledge are interconnected A previous paper in Quæstiones Informaticæ - "Laws and Techniques of Information Systems" [11] - introduced a general classification of knowledge types The present paper now extends that classification, establishes a general model of utilitarian knowledge, and derives a specific model of I S knowledge

Laws and Techniques

The academic field of study "Information Systems" is concerned with a particular kind of man-made **artefact** [3] computer based systems which produce information Information systems are similar in purpose to many other devices that people have developed over the ages, from simple hoes and clay tablets through ploughs and ledgers, to tractors and accounting machines All such inventions were motivated by the same objective, namely increased human **productivity** The prospect of ever greater output per man per hour constantly lures us on to seek more powerful artefacts However, the designs of our artefacts are critically dependent upon the knowledge base provided by subjects such as Physics, Electrical Engineering, Chemistry, Chemical Engineering, Biology, Agriculture - and **Information Systems** The previous paper showed that this dependency relationship strongly influences the desired structure of those subjects Firstly, the dependency demands that the knowledge base should include two distinct types of ideas laws and techniques

The process of designing any artefact involves a series of decisions If those decisions violate the characteristic properties of the artefact's components, then the artefact will not work as intended, and the design process will therefore be ineffective So effectiveness demands that design decision-making include inferences from **laws** – statements which describe the attributes of the various entities that are combined to form the artefact [11] For example, in deciding the appropriate curvature of a microscope's lenses, the optical designer needs to make deductions from Snell's Law of diffraction

"The sine of the angle of refraction bears a constant ratio to the angle of incidence" [7]

In designing Man's various artefacts , some of those decisions are made over and over again For efficiency of the design process, such recurring decisions need to be supported by **techniques** – statements which prescribe the steps a designer should take to reach a conclusion quickly For example, designers of optical instruments often face decisions on the spacing between lenses These decisions can be made more rapidly if one applies standard "graphical ray tracing" techniques [7] instead of Snell's Law

So for efficiency and effectiveness, the Design Process in general requires laws which describe the **operands** of design, and techniques which prescribe feasible sequences of decision-making **operations** The operations of an effective technique must be consistent with the characteristics of the operands This means that techniques should be logically connected to underlying laws [11] For example, the optical ray tracing techniques mentioned above are logical consequences of Snell's Law Similarly electrical engineering techniques of electric circuit analysis were derived form underlying laws of electricity, chemical engineering techniques of mass and energy transfer follow from basic laws of mass and energy conservation, and so on

Natural and Artificial Laws

Secondly, the dependency between Knowledge and Design demands two distinct types of laws – natural, and artificial Modern artefacts represent design "hierarchies" They consist of artificial components, which in turn are composed of natural components For example, an artesian well consists of a pump, pipes and an electric motor the motor in turn consists of copper wire and an iron frame Natural components such as copper and iron are described by "Laws of Nature" – laws that reflect their **inherent** properties On the other hand, artificial components such as motors have **contrived** properties which transcend those of their natural constituents These additional properties are described by "Laws of the Artificial" [11] For example, the natural metallic conductors in a motor are subject to Ohm's Law – a law of Nature On the other hand, an artesian well is described by overall performance formulae which are Laws of the Artificial

Again, these two types of knowledge should be connected The contrived properties of an artificial entity are functions of the inherent properties of its natural constituents That means artificial laws should be logically related to natural laws For example, the artificial laws of electric motors follow logically from natural laws of resistance and induction, the artificial laws of capacitors follow from the natural law of electrostatic force, and so on

General and Specific Laws

Thirdly, the Design-Knowledge dependency also requires a distinction – not mentioned in the previous paper – between "general" and "specific" laws Design components have two kinds of properties – general and specific – and those need to be described by corresponding types of laws For example, very many mechanical design components involve motion at sub-light velocities The set of all "sub-light motions" is described by Newton's **general** Laws of Motion However, that set contains smaller subsets of specific motions, such as rotation, oscillation and orbital motion [4] These subsets are described by **specific** laws – such as Kepler's Laws of orbital motion

If a design decision involves an operand which is a member of a specific sub-class within a general class of design components, the decision can be made more efficiently using the specific rather than the general law For example, in many orbital calculations it is easier to use Kepler's Laws rather than Newton's Laws Similarly in optical design involving "thick" lenses, the Gaussian Formulae [7] are more convenient than Snell's Law So the design process requires both general and specific laws

General and specific laws are necessarily related Suppose a large class of design operands are described by a general law Then the same law must also apply to every sub-class That means the specific law of the sub-class must be consistent with the general law So it should be possible to deduce the specific law from the general law using the specific properties as "boundary conditions" For example, Kepler's Laws are deducible from Newton's Laws, and the Gaussian Formulae can be deduced from Snell's Law

Knowledge For Design

The foregoing analysis suggests that the knowledge required by the designers of artefacts can be represented by Fig 1 At the base of the network there are general Laws of Nature Deriving from that foundation we have specific Laws of Nature, then

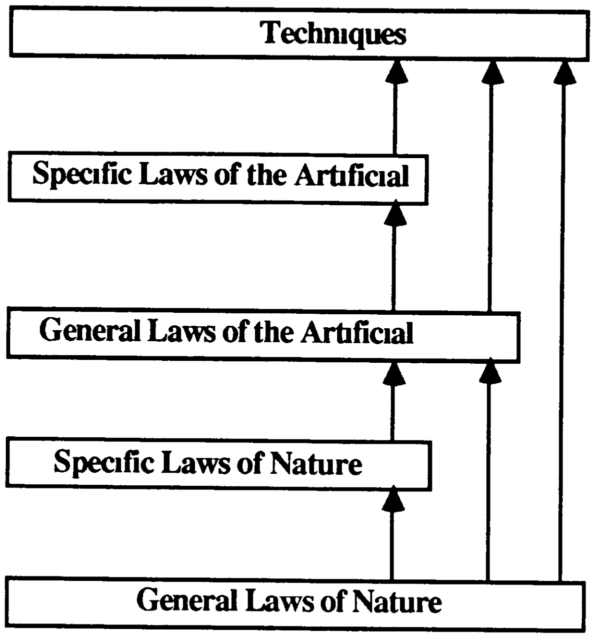

Techniques
Specific Laws of the Artificial
General Laws of the Artificial
Specific Laws of Nature
General Laws of Nature

Figure 1 Network of Design Knowledge

general as well as specific Laws of the Artificial, and finally techniques

The Subject Information Systems

The network includes laws and techniques belonging to many different fields of study Physics, Mechanical and Electrical Engineering provide laws and techniques for the design of mechanical and electrical devices Chemistry and Chemical Engineering supply laws and techniques for the design of chemical processes Biology and Agriculture furnish laws and techniques for the design of farming processes

Then there is also the subject Information Systems Its function is to support the design of informational artefacts It ought to provide laws and techniques to support obvious design activities such as configuration and network design, program and physical system design, logical system design (analysis), DP-organisation and human-interface design Furthermore, it should provide laws and techniques for information systems planning, project management and other decision-making activities whose names do not contain the word "design" The reason is that

> "Everyone designs who devises courses of action aimed at changing existing situations into preferred ones" [12]

Therefore virtually the entire subject matter of Information systems is part of the Knowledge Base for design This implies that I S laws and techniques should reflect the logical network structure of Fig 1

Knowledge About Information Systems

Thus our subject-matter should include

- laws and techniques to support decisions involving **artificial** entities such as hardware, software, systems, projects etc
- laws and techniques to support decisions involving **natural** entities such as the human element in systems and projects

For effective and efficient decision-making, the techniques should be logically connected to laws, artificial laws should be logically connected to natural laws, and specific laws should be connected to general laws

Most general laws of human behaviour, management, and computers properly fall within the ambits of other subjects – Psychology, Management Theory, Computer Science, etc Accordingly, many I S laws ought to be specific instances of general laws established in other subjects Therefore one would expect the subject to be structured like a tree I S laws should be logically derived from external laws, like a tree stem connected to ground water by its **roots** I S techniques should be logically derived from I S laws, like a tree's foliage connected to the stem by branches (Fig 2)

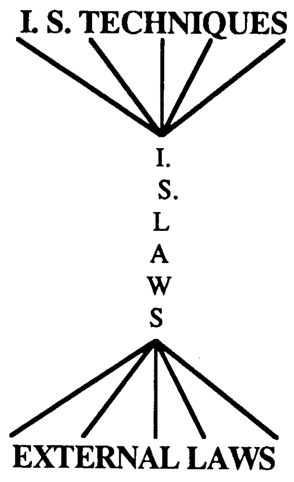


Figure 2 Tree of I.S. Knowledge

Structural Evidence

Empirical evidence of this structure can be found in two I.S. publications. Firstly, Yourdon & Constantine's "Structured Design" [14] reflects the anticipated root and stem structure. It refers to an external law, namely Miller's psychological law:

> "People can mentally ... deal with ... only about 7 ... concepts at a time" (p69).

From Miller's general natural law the book derives a specific natural law:

> "we can win if we can divide any task into independent sub-tasks" (p70).

Then from that law the book logically derives a general Law of the Artificial:

> "total systems cost will be strongly influenced by the degree of coupling between modules" (p85).

This leads on to a specific Law of the Artificial:

> "data-coupled systems have lower coupling than control-coupled systems" (p.86).

Secondly, the author's own paper, "A priority criterion for serial computer system development projects" [10] provides evidence of the expected branch structure. It presents a three-step technique for determining the relative priority of systems development projects: a) estimate parameters, b) calculate priorities, and c) rank projects in declining priority sequence. This technique has been derived mathematically from four artificial laws:

1. a system's contributions decline with age
2. a distant future contribution is less valuable than an equal contribution received in the near future
3. organisational growth increases a system's contribution
4. organisational learning enhances the system's contribution

These two publications suggest that Information systems laws and techniques can conform to the predicted tree structure. However, as indicated at the beginning, many of our publications lack the expected logical connections.

Practical Implications

The prevalence of insular techniques suggests that the subject Information systems is deficient in laws. This implies that design decisions are being made in practice with inadequate formal knowledge of the properties of design components, and are therefore likely to be ineffective. As a result, practitioners are obliged to adopt a trial and error approach.

Insular techniques are also easy to mis-apply. If a decision maker is unaware of a technique's implicit assumptions, he cannot recognise its limitations, and will use it blindly. Then if he applies the technique in a situation where the design components do not actually comply with the underlying laws, the resulting decision will be ineffective. So a costly practical learning process is necessary before one can identify situations where the technique is or isn't applicable. This may explain many an employer's preference for experience before qualifications when hiring computer personnel.

Furthermore, an insular technique is in danger of unwarranted condemnation. In the absence of explicit assumptions and logical derivation the reader may well suppose that it is intended to be universally applicable. Then if the reader has experience of a situation in which it fails, that single exception may totally invalidate the technique from the reader's point of view, even though it might be very useful in many other situation.

Conclusions

The tree model abstracts the distilled experience of researchers in Physics, Engineering, etc and transfers it to the field of Information Systems. We can benefit from that experience in three ways. Firstly, the model induces a healthy scepticism of techniques which are unsupported by logical derivation from underlying laws. Conversely, it lets us appreciate the value of those few publications which introduce laws and techniques with page after page of abstract reasoning before proceeding to concrete applications.

Secondly, it suggest ways of identifying and correcting insular techniques in our existing literature. Such techniques can be identified simply by checking their logical connections. They can then be corrected by isolating the implicit assumptions on which they are based, and developing a connecting chain of reasoning.

Finally, the model provides guidelines for future research. It recommends that we deduce I.S. techniques from underlying laws. Similarly, it suggests that we deduce I.S. laws from root laws in Psychology, Management, Computer Science, etc. Above all, it urges us to pay more attention to the **theoretical development** of our subject.

References

[1] S. Blumenthal, [1969], *Management Information Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 2.

[2] G.B. Davis, [1982], Strategies for information requirements determination, *IBM Systems Journal*, **21** (1), 5-30.

[3] P. Drucker, [1981], Work and Tools, *Ann. History of Computing*, **4** (4), 342-347.

[4] G. Fowles, [1962], *Analytical Mechanics*, Holt, Rinehart & Winston, New York.

[5] C. Gane and T. Sarson, [1979], *Structured Systems Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

[6] M.A. Jackson, [1975], *Principles of Program Design*, Academic Press, London.

[7] F. Jenkins and H. White, [1957], *Fundamentals of Optics*, McGraw-Hill, New York, 34-36, 46-48, 63-68, 119-128.

[8] M. Lundberg, G. Goldkuhl and A. Nilsson, [1981], *Information Systems Development*, Prentice-Hall, Englewood Cliffs, New Jersey.

[9] J. Martin, [1982], *Strategic Data Planning Methodologies*, Prentice-Hall, Englewood Cliffs, New Jersey.

[10] J. Mende, [1985], A priority criterion for serial computer system development projects, *S.Afr.J. Bus. Mgmt.* **15** (3), 55-60.

[11] J. Mende, [1986], Laws and Techniques of Information Systems, *Quæstiones Informaticæ*, **4** (3), 1-6.

[12] H.A. Simon, [1969], *The Sciences of the Artificial*, M.I.T. Press, Cambridge, Massachusetts, 55.

[13] V. Weinberg, [1980], *Structured Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

[14] E. Yourdon and L. Constantine, [1979], *Structured Design*, Prentice-Hall, Englewood Cliffs, New Jersey.

The Use of Colour in Raster Graphics

P J Smit
NRIMS Building, CACDS, CSIR, PO Box 395, Pretoria, 0001

Abstract

Colour computer graphics covers a wide field of applications such as games, computer aided-design, image processing and digital terrain modelling. The traditional way of using colour in raster graphics systems, the RGB colour model, is not always acceptable to the user who finds it unnatural and difficult to use. These disadvantages can be overcome by the use of colour models that provide natural interfaces, such as the HSI, HSL and CNS colour models, or a uniform colour space such as the LUV colour model. This paper describes these models and discusses their use in colour assignment and other applications.
Keywords: *Colour, colour models, uniform colour spaces, colour in raster graphics.*
Computing Review Category: *I.3.7.*

Received October 1986, Accepted January 1987

1. Introduction

The use of colour in graphics applications covers a wide field. Colour is used in games, painting programs, computer aided design, educational programs, image processing, digital terrain models (DTM's), and even as a basis for animation [1]. In many applications colour is used to enhance a picture. A typical example of this is the assignment of colours to a grey scale image to accentuate contrasts. In this case the colours should differ as much as possible. In other applications the colour assignment is used to represent some order, for example, heights in digital elevation models.

The way in which colours have to be specified for graphics hardware is not always acceptable to the user. Most colour display units use the three primary colours, red, green and blue, as the basis for the specification of colours. This RBG colour space is not "natural" to the user. For interactive applications a colour model that corresponds more closely to the way in which users think about colour may be preferable. The RGB colour space is also not uniform, that is, colours that are equidistant in the RGB colour space do not appear to be equidistant to the human eye. For example, the human eye can distinguish more shades of green than blue. Applications that use colours to represent a range of values could make good use of a uniform colour space.

Because of the disadvantages of the RGB colour space, several other colour models have been developed. The user specifies a colour by using a particular colour model, and this colour is then converted into RGB and applied to the hardware.

A study of the literature on the use of colour in raster graphics showed the lack of a comprehensive survey of these colour models and their usefulness. This paper attempts to meet this need. Four applicable colour models (in addition to the RGB model) are described and compared, and their use in colour assignment are discussed. The last part of the paper is an extract from an article by G M Murch [2], giving valuable guidelines for the effective use of colour in computer graphics.

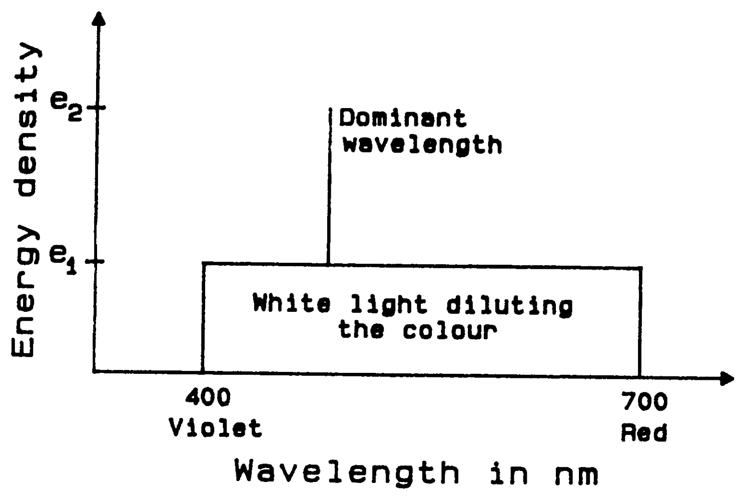
The study on the use of colour was motivated by work done on DTM's in the Computer Science Division of the National Research Institute for Mathematical Sciences, where a library of colour-handling routines was needed for the DTM package. All the colour models presented here were implemented for the Ikonas RD3000 raster graphics system. The routines were written in the SCRAP language [3] and combined in a library called IKSP (Ikonas Support Package) [4].

2. What is Colour?

This section briefly discusses some of the physiological aspects of colour. More detail may be found in [2,5].

2.1 Hue, saturation and brightness

The distinction between chromatic colours such as green and yellow and achromatic colours namely black, grey and white is that chromatic colours have a hue component while achromatic colours have none. **Hue** is the term used to distinguish between colours such as red, blue and yellow, and is an attribute of the wave length of the light rays. **Saturation** (or purity) refers to the extent to which a colour departs from a neutral grey and approaches a pure spectrum colour. A pure colour (that is, one with no white light added) has total saturation, while white light has zero saturation. **Brightness** (also



Hue $= f(\text{dominant wavelength})$

Purity $= f\left(\dfrac{\text{amount of dominant wavelength}}{\text{amount of white light}}\right) = f\left(\dfrac{e_2}{e_1}\right)$

Luminance $= f\left(\begin{array}{c}\text{amount of light}\\ \text{or total energy}\end{array}\right) = f\left(\begin{array}{c}\text{area underneath}\\ \text{the curve}\end{array}\right)$

Note: $e_1 = e_2 \Rightarrow \text{purity} = 0\%$

$e_1 = 0 \Rightarrow \text{purity} = 100\%$

Figure 1 Physical interpretation of hue, purity and luminance (adapted from [6])

called **luminance, lightness** or **intensity**) corresponds to the total energy or amount of light in a colour.

In the literature different meanings are attached to the terms lightness, brightness, luminance and intensity. Normally, brightness and lightness refer to objects that are light reflectors, while luminance and intensity are used for light sources. The lightness of an object depends on the amount of light reflected by the object and is a property of the object itself. The brightness of an object depends on the amount of light illuminating the object. If the amount of light illuminating an object is increased, the brightness of the object will increase, but its lightness will stay the same. The luminance of a light source refers to the amount of light energy emitted from the source per unit area, while intensity refers to a hardware function that can be set programmatically or manually.

It must be noted that the terms intensity and lightness used for the HSI and HSL models are used as defined in the descriptions of these models and not in the sense described above. In section 5, **Guidelines for effective colour use**, the other terms for brightness are used as defined above.

2.2 Physiological aspects of colour

The colour seen by human beings is the result of the physical properties of light entering their eyes. The wavelength of visible light ranges from 400 nm (violet) to 700 nm (red). Figure 1 shows the relation between the physical properties of light and the hue, purity and luminance of a colour.

The retina of the human eye contains two types of sensors, called rods and cones. The rods are used for night vision, while the cones contain photopigments that translate light wavelengths into colour sensations. These photopigments are sensitive to red, green or blue light. Because of the variation in the distribution of both photopigments and cones across the retina, we have different response characteristics for each colour (see figure 2) [6]. The combination of these responses results in the sensation of a specific colour.

It is important to note the difference between the colour on a raster graphics display unit and that on a surface such as a painting. The colour display unit uses additive colour mixing, while the colours in a painting are obtained by subtractive colour mixing.

Additive colour mixing starts with black and obtains a colour by mixing coloured lights. Red, green and blue are well known examples of the so-called additive primaries. Their individual contributions are mixed together to form an additive mixture with a certain colour. If the three primaries are mixed together in equal proportions, white light is obtained.

Subtractive colour mixing starts with white light and obtains a colour by subtracting colours from the white light. For example, a piece of paper will be blue if all the colours other than blue are subtracted from the white light illuminating it. Examples of the so-called subtractive primaries are cyan, magenta and yellow, which are the complementary colours to red, green and blue respectively. The relationship

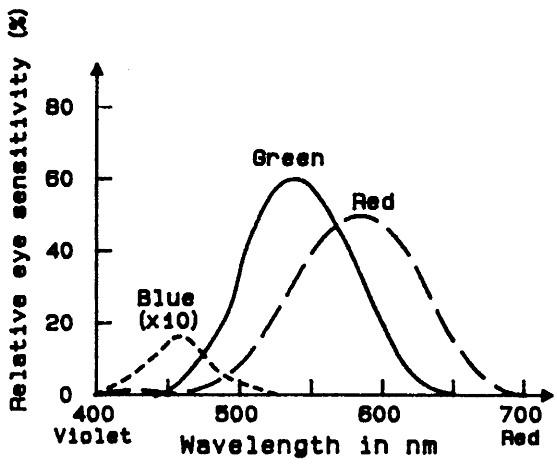


Figure 2 Response characteristics of the human eye (adapted from [6])

between these additive and subtractive primaries can be expressed by

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

where each primary ranges between 0 and 1. The colours obtained by mixing additive or subtractive primaries are illustrated in figure 3.

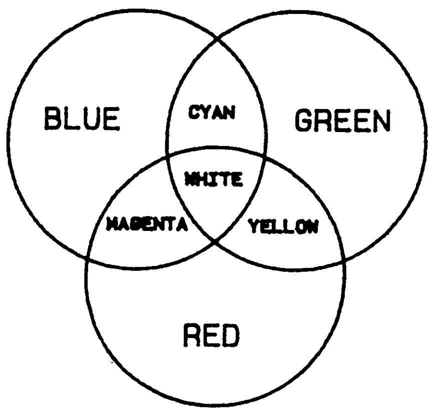


Figure 3a Additive colour mixing with red, green and blue as primaries

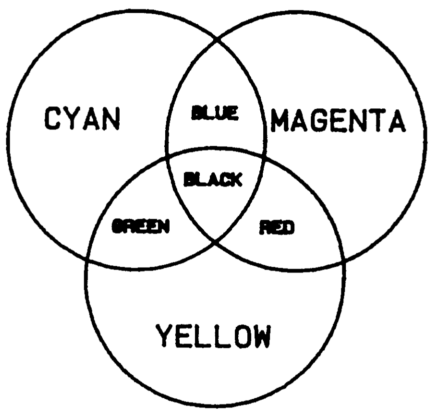


Figure 3b Subtractive colour mixing with cyan, magenta and yellow as primaries

3. Colour Models

In this section the RGB, HSI, HSL, CNS and LUV colour models are outlined and compared. A general description of each model is given, together with the relevant conversion routines and a discussion of the model's use.

It is interesting to note that the HSI conversion routines given here and those given in [6,7,8] are equivalent, although they seem to differ with regard to the definition of hue. The conversion routines for HSL given here and in [6,8,9] are also equivalent (except that [8] differs with regard to the definition for hue). Finally, the seemingly different definitions given by [6] for the hue component of HSI and HSL are identical.

Throughout this section it is assumed that
max = the maximum value of {R,G,B}
mid = the middle value of {R,G,B}
min = the minimum value of {R,G,B}.

3.1 RGB Model

3.1.1 General description

A colour raster graphics display unit is coated with phosphor that emits light when struck by the electron beam of the display unit. Three different types of phosphor, each emitting one of the three primary colours, are used. These primary colours, red, green and blue, (R,G,B) can be combined with positive weights to obtain a large range of colours. However, some visible colours cannot be defined in terms of positive weights for the (R,G,B) primaries. This is why the Commission Internationale L'Eclairage (CIE) in 1931 defined the three primary colours (X,Y,Z) as the international standard for specifying colour (see [5]). These primaries can be combined with positive weights to define all the light sensations that we perceive with our eyes.

The geometric model associated with the RGB model is a cube (see figure 4) with a primary along each of the three axes. The value of each primary ranges between 0 and 1. The nine corners of the cube correspond to black, white, the three additive primaries, and the three subtractive primaries.

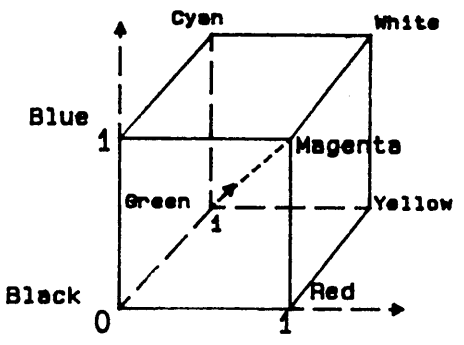


Figure 4 The RGB colour cube

3.1.2 Converting between XYZ and RGB

A set of routines is needed to convert between the (X,Y,Z) primaries of the CIE and the (R,G,B) primaries of colour display units.

LET

P = matrix of chromaticities of phosphors used in the display unit

$$= \begin{bmatrix} x_r & x_g & x_b \\ y_r & y_g & y_b \\ 1-(x_r+y_r) & 1-(x_g+y_g) & 1-(x_b+y_b) \end{bmatrix}$$

P^{-1} = inverse of matrix P

$$V = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} = \text{vector of tristimulus values of CIE standard illuminant divided by 100}$$

$$\begin{bmatrix} c_r \\ c_g \\ c_b \end{bmatrix} = P^{-1}V$$

THEN

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = P \begin{bmatrix} c_rR \\ c_gG \\ c_bB \end{bmatrix}$$

$$\begin{aligned} X &= c_rx_rR + c_gx_gG + c_bx_bB \\ \text{giving } Y &= c_ry_rR + c_gy_gG + c_by_bB \\ Z &= c_rz_rR + c_gz_gG + c_bz_bB \end{aligned}$$

The value of P depends on the colour display unit (see [5] for more detail). The standard for chromaticities of colour television were set by the U.S. National Television System Committee (NTSC) in 1953. According to this standard,

$$P = P1 = \begin{bmatrix} 0.67 & 0.21 & 0.14 \\ 0.33 & 0.71 & 0.08 \\ 0.00 & 0.08 & 0.78 \end{bmatrix}$$

However, the chromaticities of the phosphors now commonly used in colour television deviate somewhat from the NTSC standard:

$$P = P2 = \begin{bmatrix} 0.68 & 0.28 & 0.15 \\ 0.32 & 0.60 & 0.07 \\ 0.00 & 0.12 & 0.78 \end{bmatrix}$$

The values used for the vector V depend on the light source illuminating the colour display unit. Normally, this is taken as one of the CIE standard light sources corresponding to average daylight, namely C or D65. The current standard practice is to use the D65 standard illuminant. Tables of the so-called tristimulus values for CIE standard illuminants are given in [5] for the CIE 1931 Standard Colorimetric Observer and for the CIE 1964 Supplementary Standard Colorimetric Observer. Preference should be given to the 1964 data when areas of the same colour are large (that is, more than 2 cm in diameter). The 1931 data are more suitable for smaller colour areas (that is, less than 2 cm in diameter). See [5] for a detailed discussion of standard light sources.

The tristimulus values, divided by 100, of the C and D65 standard illuminants are as follows.

$$V = V1 = \begin{bmatrix} 0.98041 \\ 1.0 \\ 1.18103 \end{bmatrix}, \quad V = V2 = \begin{bmatrix} 0.95017 \\ 1.0 \\ 1.08813 \end{bmatrix}$$

C Illuminant 1931 Observer — D65 Illuminant 1931 Observer

$$V = V3 = \begin{bmatrix} 0.97298 \\ 1.0 \\ 1.16137 \end{bmatrix}, \quad V = V4 = \begin{bmatrix} 0.94825 \\ 1.0 \\ 1.07381 \end{bmatrix}$$

C Illuminant 1964 Observer — D65 Illuminant 1964 Observer

If the second set of values for P (P = P2) is used, as well as the 1964 data for D65 (V = V4), the following is obtained:

$$\begin{aligned} X &= 0.437509R + 0.331566G + 0.179175B \\ Y &= 0.205887R + 0.710498G + 0.0836149B \\ Z &= 0.1421G + 0.931709B \end{aligned}$$

The inverse transformation is given by

$$\begin{aligned} R &= 2.87574X - 1.25391Y - 0.440496Z \\ G &= -0.848557X + 1.80318Y + 0.00135981Z \\ B &= 0.129418X - 0.275013Y + 1.07309Z \end{aligned}$$

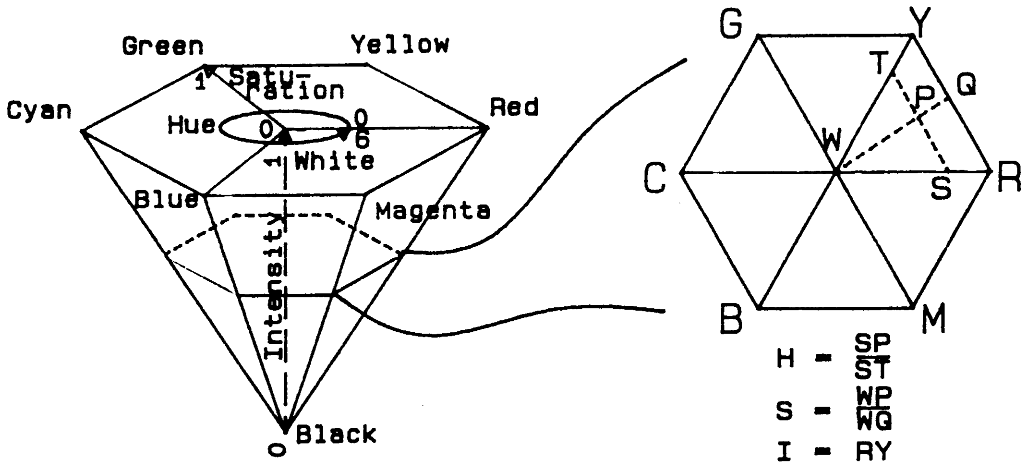


Figure 5 The HSI colour model. H, S, and I are defined using the geometric model

3.1.3 Use of the RGB model

The shortcomings of the RGB colour model, namely its lack of naturalness and uniformity, have already been mentioned.

However, many applications still use this model. The RGB model, being hardware oriented, needs no time-consuming conversions. Furthermore, there is a considerable amount of knowledge available on the eye's response and sensitivity to colours specified as (R,G,B) triples. In some instances interpolation done in RGB space may still be preferred. For example, the shadow series for a colour illuminated by a single light source is determined by linear interpolation between the particular colour and black in RGB space [8].

3.2 HSI Model

3.2.1 General description

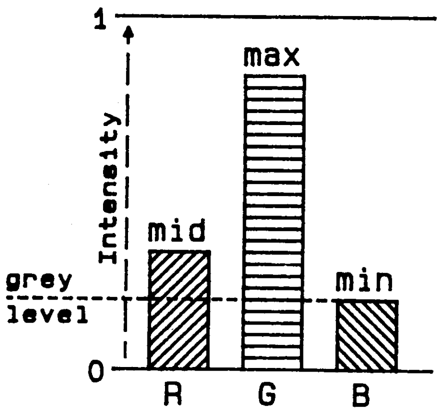
The HSI (hue, saturation and intensity) model [6,7,8] is intended to appear more natural to the user, as it is based on the intuitive appeal of tint, shade and tone used by the artist. In the literature different names are used for the three components, but here the terms hue, saturation and intensity are used.

The subspace within which the colour model is defined is a hexcone (figure 5). A formal derivation of the model can be found in [7]. Only a general description is given here.

The vertical axis represents the intensity of the colour, where intensity may be seen as the distance from black. On the vertical axis itself the colour will be achromatic, going from black (at I = 0) through grey to white (at I = 1). The hexcone is defined such that the length of a side of a hexagon disk (RY, for example) is the same as the value of I for that disk.

The saturation (S) of a colour is defined as a ratio depending on the horizontal distance from the vertical axis of the hexcone. In figure 5 the saturation of the colour at P will be the ratio WP/WQ. The value of S is 0 at the vertical axis and 1 on the triangular sides of the hexcone.

The third component defining a colour is its hue. In the hexcone model the hue depends on the angle around the vertical axis, starting at red and moving in an anticlockwise direction (by convention). The hue value ranges between 0 (which is red) and 6 (which is again red). In figure 5 the hue of the colour at P will be the ratio SP/ST.



$$I = \text{max}$$

$$S = \frac{\text{max} - \text{min}}{\text{max}}$$

$$H = f(h), \text{ where } h = \frac{\text{mid} - \text{min}}{\text{max} - \text{min}}$$

Figure 6 Bar representation of colour components and their relation to hue (H), saturation (S) and intensity (I)

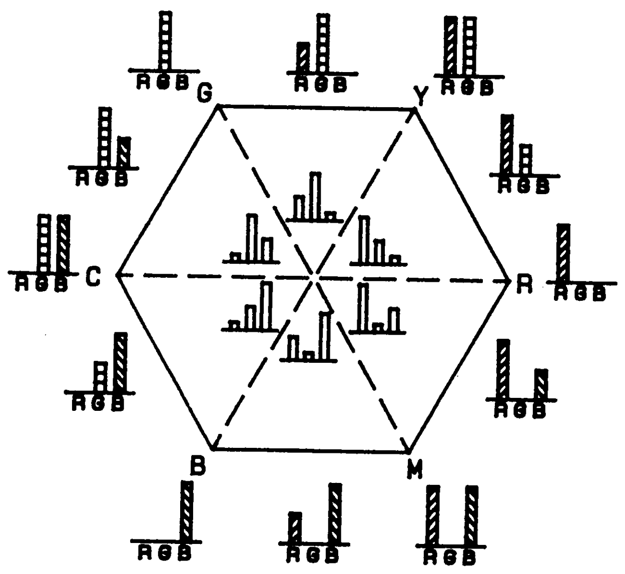
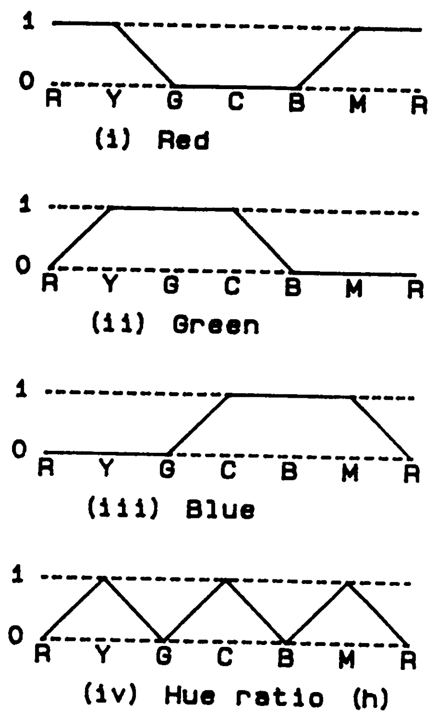
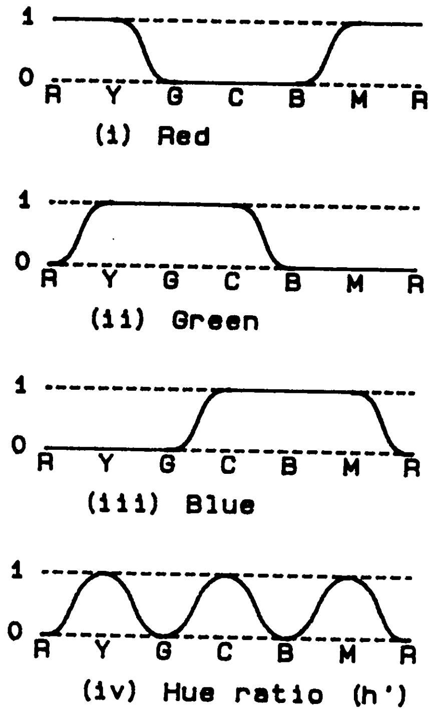


Fig 7
HSI colour hexagon showing the R-G-B ratios.The pure colours are on the circumference and the de-saturated colours within the hexagon.



Linear variation of the hue components and the hue ratio (h).

Fig 8a



Sinusoidal variation of the hue components and the hue ratio (h'), using $h' = (1 - \cos(\pi h))/2$

Fig 8b

3.2.2 RGB to HSI conversion

In the RGB colour space a colour can be represented by three bars (figure 6). The colour is obtained by mixing R, G and B in the proportions implied by the height of the three bars. The intensity (I) of the colour is then defined as the height of the bar: I = max.

The height of the smallest bar is min, and the RGB triple (min, min, min) is the grey which desaturates the colour. The saturation of the colour is defined as the ratio of the non-grey part of the colour to its whole:

S = (max – min)/max. For the case where R = G = B = 0, S is also defined to be 0.

Subtraction of the grey level (min, min, min) from the colour leads to the observation that a hue is determined by two primary colours only, namely the maximum and middle values of R, G and B. Hue H therefore depends on the ratio h, where h = (mid – min)/(max – min).

In the hexcone model, the hue is a modular function, cycling through red, yellow, green, cyan, blue and magenta. This must be mapped onto a hexagon plane of the hexcone model. Figure 7 shows how the R, G and B components vary in the different sectants. The change in value for each component is given separately in figure 8a. The hue ratio h, which oscillates between 0 and 1, is also depicted.

The hue of a colour is defined as a different function of h for each sectant (see figure 9). Thus the value of H ranges between 0 and 6. Because this function is noncontinuous and the human system tends to enhance these variations, a hue series will exhibit Mach banding. This may be overcome by using a sinusoidal function h', where h' = (1 – cos(πh))/2 [8]. Figure 8b shows the values of each component when this function is used.

Segment	Hue ratio (h)	Definition of Hue (H)	Range of H
R-Y	0 - 1	h	0 - 1
Y-G	1 - 0	2 – h	1 - 2
G-C	0 - 1	2 + h	2 - 3
C-B	1 - 0	4 – h	3 - 4
B-M	0 - 1	4 + h	4 - 5
M-R	1 - 0	6 – h	5 - 6

Figure 9 Definition of the hue component (H) in terms of the hue ratio (h)

If S = 0 (that is, max = min), the value of H is undefined. For practical purposes, however, H is set to 0 (red) whenever S = 0. This gives us the following:

GIVEN: R, G, and B, each in [0, 1]
DESIRED: H in [0, 6), S and I in [0, 1]
FORMULA:
I = max

$$S = \begin{cases} \dfrac{\max - \min}{\max} & \text{if } \max \neq 0 \\ 0 & \text{if } \max = 0 \end{cases}$$

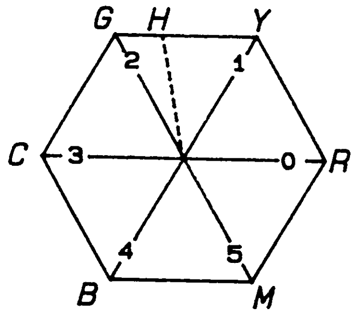
$$H = \begin{cases} 0 & \text{if } S = 0 \\ h & \text{if } R \geq G \geq B \\ 2 - h & \text{if } G > R \geq B \\ 2 + h & \text{if } G \geq B > R \\ 4 - h & \text{if } B > G > R \\ 4 + h & \text{if } B > R \geq G \\ 6 - h & \text{if } R \geq B > G \end{cases}$$

where

$$h = \begin{cases} \dfrac{\text{mid} - \min}{\max - \min} & \text{if linear hue variation} \\ \dfrac{1}{2}\left[1 - \cos\left(\pi\dfrac{\text{mid} - \min}{\max - \min}\right)\right] & \\ \text{if sinusoidal hue variation} & \end{cases}$$

3.2.3 HSI to RGB conversion

This is simply the inverse of the RGB to HSI conversion.



GIVEN: H in [0, 6), S and I in [0, 1]
DESIRED: R, G, and B, each in [0, 1]
FORMULA:
max = I
min = max (1 – S)
mid = min + fract(max – min)
where

$$\text{fract} = \frac{\text{distance from nearest primary (e.g. } GH)}{\text{length of segment (e.g. } GY)}$$

$$= \begin{cases} h = \text{ABS}\left(2\left\lfloor \dfrac{H}{2} + \dfrac{1}{2}\right\rfloor - H\right) & \\ \qquad \text{if linear hue variation} & \\ \dfrac{1}{\pi}\arccos(1 - 2 \times h) & \\ \qquad \text{if sinusoidal hue variation} & \end{cases}$$

```
SELECT ⌊H⌋ FROM
CASE 0:  (R,G,B) ← (max, mid, min)   {RY}
CASE 1:  (R,G,B) ← (mid, max, min)   {YG}
CASE 2:  (R,G,B) ← (min, max, mid)   {GC}
CASE 3:  (R,G,B) ← (min, mid, max)   {CB}
CASE 4:  (R,G,B) ← (mid, min, max)   {BM}
CASE 5:  (R,G,B) ← (max, min, mid)   {MR}
ENDS
```

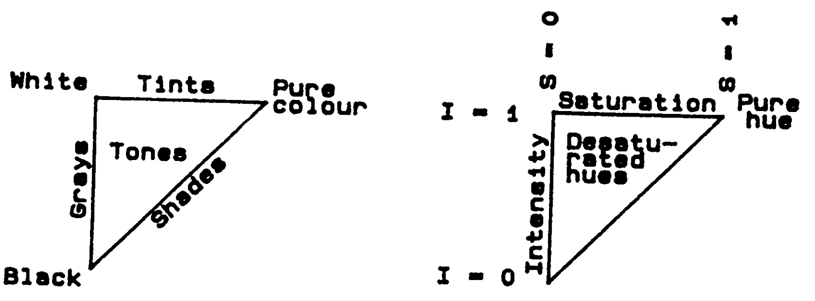


Figure 10 Correspondence between the artist's tint, tone and shade and the HSI model's hue, saturation and intensity

3.2.4 Use of the HSI model

When implemented, the routines to convert between HSI and RGB must make provision for the fact that most hardware requires non-negative integer parameters. As the use of integers causes a loss of accuracy, a conversion from RGB to HSI and back to RGB will not always return the same RGB values. However, the resolution of the hardware is normally such that this error is invisible to the human eye.

It must also be noted that, because of the nature of the HSI model, more than one HSI value will be mapped onto the same RGB value. On the other hand, this RGB value will be mapped onto one HSI value only. The convention that H = 0 when S = 0 may be changed to fit the application [7].

The biggest advantage of the HSI model is its intuitive appeal to the user. The correspondence between hue, saturation and intensity and the artist's use of tint, shade and tone (see figure 10) makes the HSI model very suitable for painting on a raster screen. This model has also been used in a program teaching colour theory to art students [10].

In [7,11], an example of so-called "tint painting" is given. Here the "tint" (hue and saturation) of a picture is changed, but the grey level (intensity) is to stay the same. This is done by the conversion of the RGB values of the pixels underneath the "paint brush" to HSI and the extraction of their I values. These I values are then combined with the H and S of the new tint and converted to obtain a new RGB value for the pixels.

From the tint painting example it is clear that the conversion must be fast. To this end HSI is very suitable, because it is possible to do the conversion without using floating point arithmetic.

In the HSI model the pure, maximally saturated hues are at S = 1 and I = 1. This is an advantage (when compared with HSL) if potentiometers are used to specify the colour model parameters.

A disadvantage of the HSI model is that it is not possible to go, for example, from black through dark green, green and light green to white by changing only one parameter (as can be done with HSL). Both S and I have to be changed. An advantage (over RGB, for example) is that it is possible to go through all the hues at a certain saturation and intensity by changing only the value of H.

The HSI colour model is not a uniform colour space. Colours that are equidistant in the model are not necessarily perceived as being equidistant by the human eye. This is especially noticeable in the green region of the colour space.

The sinusoidal hue variation attempts to improve the uniformity of the hue component. The effect of the sinusoidal function is that the hues are placed nearer to one another at the red, yellow, green, cyan, blue and magenta regions, and spaced wider in the red-yellow, yellow-green, green-cyan, cyan-blue, blue-magenta and magenta-red regions. However, whether one compares the distances in the uniform LUV colour space or the visual results, there does not seem to be a real improvement.

When interpolation between two colours in a colour space is done, the intermediate points differ for each colour space. In some instances of interpolation, the HSI colour space may be preferred [8]. Interpolating between a colour and an unsaturated blue in HSI simulates the effect of atmospheric scattering more closely than similar interpolation in (say) RGB [12]. HSI allows one to keep the saturation and intensity constant while interpolation is done between two hues, whereas this is difficult to achieve in RGB space.

The HSI model can be used for colour assignment [13] and will give better results than the RGB model in most cases. In 4.2 this topic is discussed further.

3.3. HSL Model

3.3.1 General description

The HSL (hue, saturation, lightness) colour model [6,8,9] is used by Tektronix Inc., and is similar to the HSI model. Lightness (L) is defined in such a way that it is possible to go from black through green to white by only changing the value of L.

The HSL has a double hexcone as colour subspace. The hexcone of the HSI colour model is deformed into a double hexcone by "pulling" the white upwards (see figure 11). The representation of hue and saturation in the hexcone is the same as in the

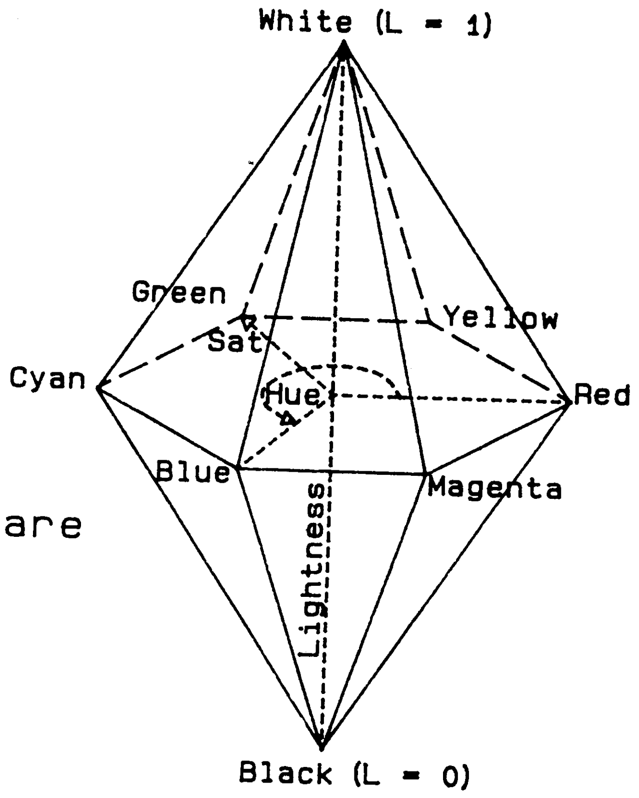


Fig 11

HSL colour model. The pure colours are at L = 0.5.

HSI model. The lightness is still represented by the vertical axis, with black at L = 0 and white at L = 1, but the fully saturated colours are at L = 0.5. Note that a fully saturated colour at L = 0.5 and S = 1 in the HSL model will have I = 1 (not I = 0.5) and S = 1 in the HSI model.

3.3.2 RGB to HSL conversion

In the HSL colour model the definition of the hue of a colour is exactly the same as in the HSI colour model. As stated above, the definition of L is changed (from the definition of I) to form a double hexcone. The definition of S is adjusted accordingly.

GIVEN: R, G, and B, each in [0, 1]

DESIRED: H in [0, 6), S and L in [0, 1]

FORMULA:

$$L = \frac{\max + \min}{2}$$

$$S = \begin{cases} \dfrac{\max - \min}{\max + \min} & \text{if } L \le 0.5 \\ \dfrac{\max - \min}{2 - \max - \min} & \text{if } L > 0.5 \\ 0 & \text{if } L = 0 \text{ or } L = 1 \end{cases}$$

$$H = \begin{cases} 0 & \text{if } S = 0 \\ h & \text{if } R \ge G \ge B \\ 2 - h & \text{if } G > R \ge B \\ 2 + h & \text{if } G \ge B > R \\ 4 - h & \text{if } B > G > R \\ 4 + h & \text{if } B > R \ge G \\ 6 - h & \text{if } R \ge B > G \end{cases}$$

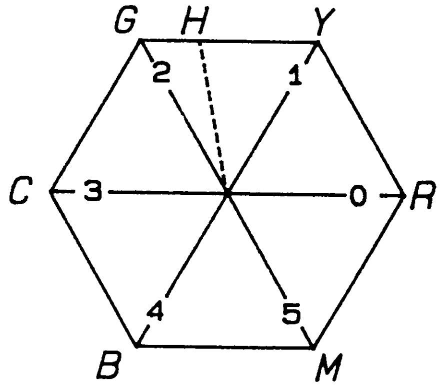
where

$$h = \begin{cases} \dfrac{\text{mid} - \min}{\max - \min} & \text{if linear hue variation} \\ \dfrac{1}{2}\left[1 - \cos\left(\pi\dfrac{\text{mid} - \min}{\max - \min}\right)\right] & \text{if sinusoidal hue variation} \end{cases}$$

Note that L is now halfway between max and min. The denominator of S therefore includes both max and min (in contrast to the HSI definition of S). If S = 1 on the lower half of the hexcone (L < 0.5), at least one of R, G or B is 0. If S = 1 on the upper half of the hexcone (L > 0.5), at least one of R, G or B is 1.

3.3.3 HSL to RGB conversion

This is simply the inverse of the RGB to HSL conversion.



```
GIVEN: H in [0, 6), S and L in [0, 1]
DESIRED: R, G, and B, each in [0, 1]
FORMULA:
max = L + L × S                  if L ≤ 0.5
      L + S − L × S              if L > 0.5
min = 2 × L − max
mid = min + fract(max − min)
where
```

$$\text{fract} = \frac{\text{distance from nearest primary (e.g. } GH)}{\text{length of segment (e.g. } GY)} = \begin{cases} h = \text{ABS}\left(2\left\lfloor \frac{H}{2} + \frac{1}{2}\right\rfloor - H\right) & \text{if linear hue variation} \\ \frac{1}{\pi}\arccos(1 - 2 \times h) & \text{if sinusoidal hue variation} \end{cases}$$

```
SELECT ⌊H⌋ FROM
CASE 0:  (R,G,B) ← (max, mid, min)      {RY}
CASE 1:  (R,G,B) ← (mid, max, min)      {YG}
CASE 2:  (R,G,B) ← (min, max, mid)      {GC}
CASE 3:  (R,G,B) ← (min, mid, max)      {CB}
CASE 4:  (R,G,B) ← (mid, min, max)      {BM}
CASE 5:  (R,G,B) ← (max, min, mid)      {MR}
ENDS
```

3.3.4 Use of the HSL model

The HSL conversion routines have the same accuracy problems as the HSI routines when integer parameters are used. Also more than one HSL value is mapped onto the same RGB value. As with HSI, the convention that H = 0 when S = 0 can be changed to fit the application.

The advantage of the HSL model (as with HSI) is its intuitive appeal to the user. It has the "natural" components of hue, saturation and lightness. An improvement on HSI is that a given tint (a H and S value) can be varied from almost black to almost white by only the L value being changed. This corresponds to our natural colour language, since we talk of dark green, green and light green. This is probably the reason why the HSL model is used by Tektronix and the CORE system [9]. For the same reason the HSL model is used as the basis of the CNS model, where colour is specified in natural language (see 3.4).

The HSL model can be used in painting routines in a way similar to the HSI model. A picture may be "painted" darker or lighter by only the L value being changed. The RGB value of a pixel underneath the paint brush is converted to HSL, the L value is replaced by the L value of the paint and the pixel is given the corresponding new RGB value. The speed of this operation should be adequate because the HSL conversion is simple and straightforward.

The advantage of the HSL model can also be a disadvantage. In some applications it would be better if the fully saturated hues were at L = 1 and not at L = 0.5. An example is when potentiometers are used to specify the parameters of HSL.

The hue component in HSL and HSI is exactly the same. The sinusoidal hue variation effects no noticeable improvement, while interpolation that keeps the saturation and lightness constant and only changes the hue is possible. Colour assignment (see 4.3) can be done in a similar way as for the HSI model.

The HSL colour space is not a uniform colour space. Uniform steps in L, for example, do not appear to be uniform to the human eye.

3.4. CNS Model

3.4.1 General description

The Colour Naming System (CNS) is not a colour model in its own right, but a naming system built upon the HSL colour model [14]. It allows the specification of colours by use of their "natural" English names. A common English term is used to describe each of the lightness, saturation and hue components of a colour.

There are five possibilities for the lightness component, namely **very dark**, **dark**, **medium**, **light** and **very light**. If this component is not specified, **medium** is used. Four saturation levels are possible, namely **greyish**, **moderate**, **strong** and **vivid**, with **vivid** being the default term.

Hue names are formed from seven generic hues, namely **red**, **orange**, **brown**, **yellow**, **green**, **blue** and **purple**. These terms, together with **black**, **white** and **grey** constitute the basic colour terms in English; **pink** is specified as **light red**, which is easily understood. Chromatic hue names are formed by the combination of the generic hue names that are adjacent on the HSL hue circle. 'Half-way' hues (for example, **yellow-green**) or 'quarter-way' hues (for example **yellowish green**) may be formed. The yellow-green hue will be half-way between yellow and green on the hue circle and yellowish green will be half-way between yellow-green and green.

Achromatic hue names are formed by black, white and grey, the latter being formed together with a lightness term (for example, light grey). Figures 12 and 13 illustrate the syntax of the CNS model.

3.4.2 CNS to RGB conversion

A colour specified in CNS notation is first converted to HSL, and then the HSL to RGB conversion routine is used. The conversion to HSL is done by the mapping of the CNS lightness and saturation terms onto corresponding L and S values of the HSL colour model. Similarly, the CNS generic hues are mapped onto corresponding H values of the model. The 'half-way' and 'quarter-way' hues are mapped onto H values half-way and quarter-way between the generic hues as shown in figure 13.

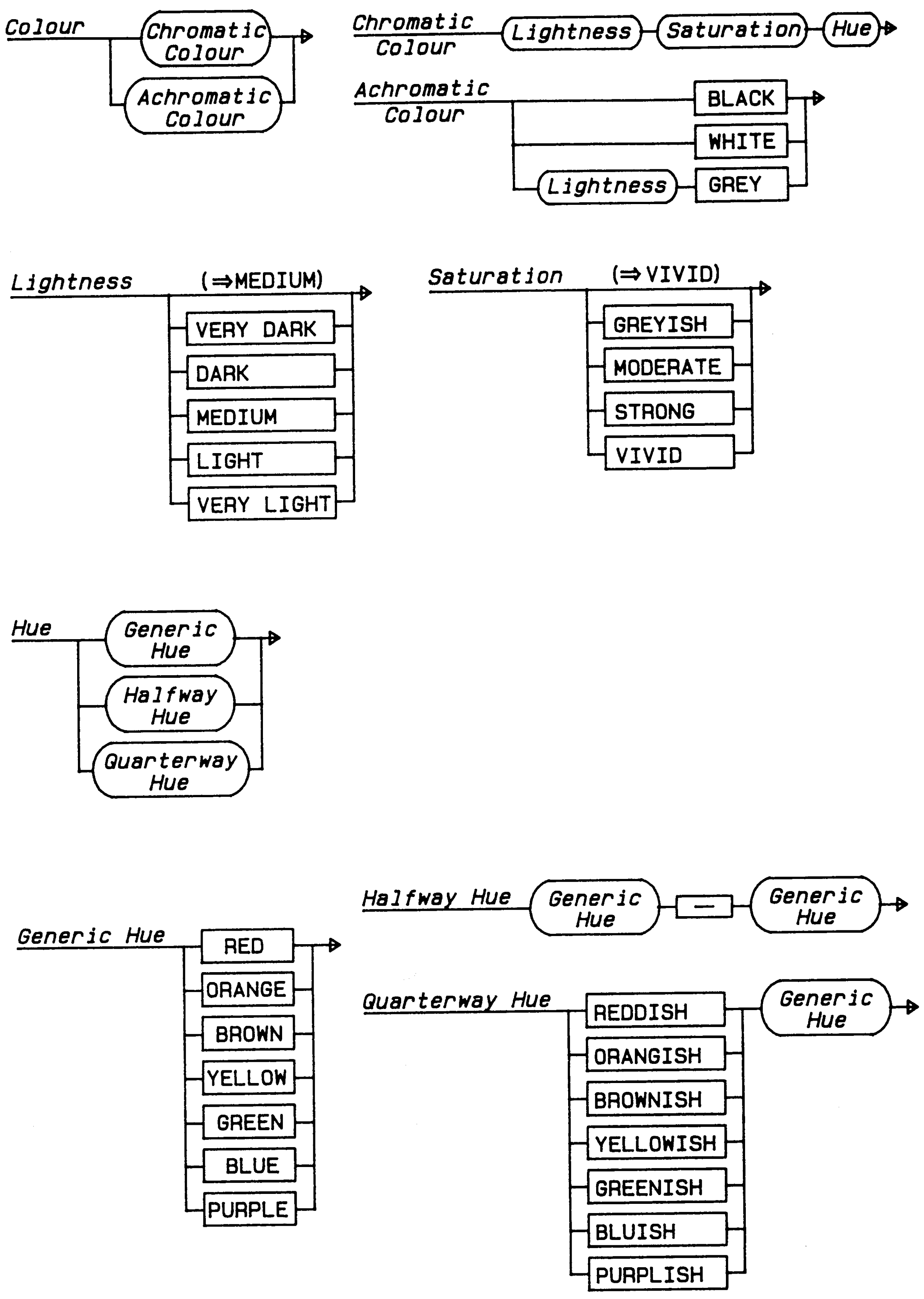
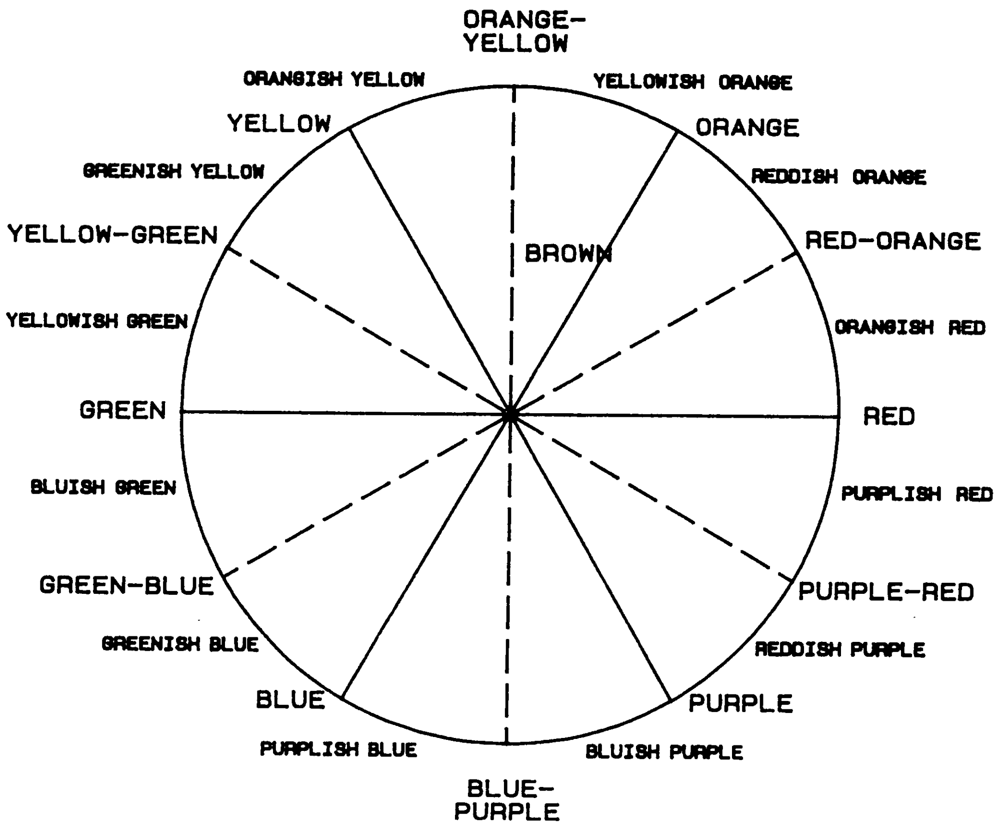


Fig 12 CNS syntax



(a) Chromatic hue names

GREEN
LIGHT BLUE
LIGHT MODERATE GREEN
REDDISH ORANGE
VERY LIGHT VIVID BLUE-PURPLE
BLACK
DARK GREY

(b) Examples

Fig 13 CNS syntax examples

3.4.3 RGB to CNS conversion

The conversion from RGB to CNS is the inverse of the CNS to RGB conversion. First a conversion from RGB to HSL and then a mapping from HSL to the CNS syntax is done. Because of the coarse resolution of CNS, many HSL colours will be mapped onto the same CNS colour as, for example, only five terms for lightness are possible in CNS. If a value is between two terms, the term that is the nearest to the actual value is selected.

3.4.4 Use of the CNS Model

The advantage of the CNS model is the ease with which it can be used. Colours are specified by using their English names and not by giving numeric values. The disadvantage is the coarse resolution imposed by the limited number of English terms used. In spite of this coarse resolution, a study has shown the superiority of the CNS model over the RGB and HSI models in the identification of colours [14].

As soon as the user requires a finer resolution for the specification of colours, the CNS model is unable to cope. A solution is to use the HSL model for finer resolution, since the CNS model uses the HSL model as basis. It remains difficult, however, to specify a range of colours in CNS (for example, all the colours from dark green to light green). This shows that the CNS model's use is limited to the specification and/or identification of one colour at a time.

Another disadvantage of the CNS model is in its implementation. It is difficult to calibrate the model, that is, to specify the values to be used for all the lightness and saturation terms and for brown. For example, a set of lightness values that is acceptable for green is not necessarily acceptable for blue.

3.5. LUV Model

3.5.1 General description

The LUV colour model [5,15,17] differs from the other colour models in that it is not intended to make colour specification more "natural" to the user. Its aim is rather to provide a uniform colour space, by which is meant a colour space in which differences in the human perception of colours correspond approximately to Euclidean distances.

The LUV model (see figure 14) was developed on an experimental basis and does not have a straightforward natural interpretation such as the HSI model. Its three components are called L, U and V. L gives the luminance of a colour and is similar to I of HSI and L of HSL. The U component gives the chromaticity variation approximately from green to red, and V gives the chromaticity variation approximately from blue to yellow.

For the LUV colour space to appear to be uniform to the human visual system, environmental factors, such as the phosphors used for the colour display unit and the properties of the light source illuminating the display unit, must be taken into account. Since this is a science in itself, only the formulas used are given here. Reference [5] gives an

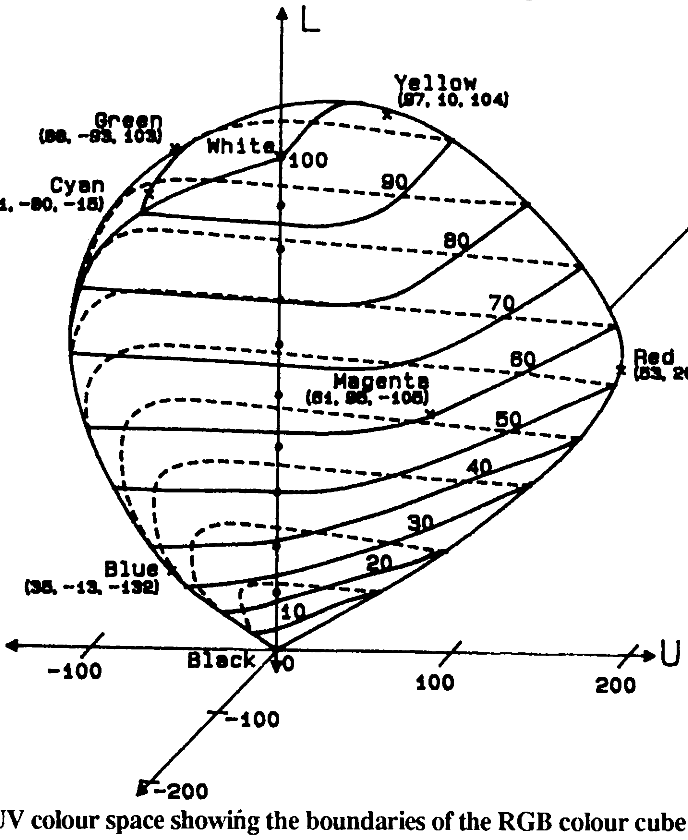


Figure 14 LUV colour space showing the boundaries of the RGB colour cube and the position of the basic colours (adapted from [5])

excellent exposition on this subject, while [16] discusses the use of uniform colour spaces for computer graphics.

3.5.2 RGB to LUV conversion

Before the formulas are discussed, it must be noted that the values of LUV are not in the [0,1] range. Each component has a different range, depending on the values chosen for the matrix P and the vector V. If P2 and V4 are used (see 3.1.2), L will range between 0 and 100, U between –93 and 204, and V between –132 and 104. The ranges could be changed to be positive integers, without loss of uniformity, by a constant being added. To improve on the resolution, each component can be multiplied by the same constant. This will still preserve uniformity.

GIVEN: R, G, and B, each in [0, 1]

DESIRED: L in [0, 100], U in [–93, 204], and V in [–132, 104], using P = P2 and V = V2 (see 3.1.2)

FORMULA:

LET

$$X = 0.437509R + 0.331566G + 0.179175B$$
$$Y = 0.205887R + 0.710498G + 0.0836149B$$
$$Z = 0.1421G + 0.931709B$$

$$u = \frac{4X}{X + 15Y + 3Z} \qquad v = \frac{9Y}{X + 15Y + 3Z}$$
$$u_0 = \frac{4X_0}{X_0 + 15Y_0 + 3Z_0} = 0.1978645$$
$$v_0 = \frac{9Y_0}{X_0 + 15Y_0 + 3Z_0} = 0.4694914$$
$$Y_0 = 1$$

THEN

$$L = 116\left(\frac{Y}{Y_0}\right)^{\frac{1}{3}} - 16 \quad \text{if} \frac{Y}{Y_0} > 0.01$$
$$U = 13L(u - u_0)$$
$$V = 13L(v - v_0)$$

3.5.3 LUV to RGB conversion

The LUV to RGB conversion is simply the inverse of the RGB to LUV conversion. It must be noted, however, that not all the LUV values within the given ranges can be converted to be valid (that is in the [0,1] range) RGB values. This is due to the form of the RGB colour cube in the LUV colour space (see figure 14).

GIVEN: L in [0, 100], U in [–93, 204], and V in [–132, 104], using P = P2 and V = V2 (see 3.1.2)

DESIRED: R, G, and B, each in [0, 1]

FORMULA:

LET

$$Y = Y_0\left(\frac{L + 16}{116}\right)^3$$
$$X = \frac{9(U + 13u_0L)}{4(V + 13v_0L)}Y$$
$$Z = \left(\frac{39L}{V + 13v_0L} - 5\right)Y - \frac{X}{3}$$

where

$$Y_0 = 1$$
$$u_0 = \frac{4X_0}{X_0 + 15Y_0 + 3Z_0} = 0.1978645$$
$$v_0 = \frac{9Y_0}{X_0 + 15Y_0 + 3Z_0} = 0.4694914$$

THEN

$$R = 2.87574X - 1.25391Y - 0.440496Z$$
$$G = -0.848557X + 1.80318Y + 0.00135981Z$$
$$B = 0.129418X - 0.275013Y + 1.07309Z$$

NOTE:

Input is illegal if one of the following is false:

1. $L = 0 \Rightarrow U, V, R, G, B, = 0$
2. $V + 13v_0L = 0 \Rightarrow L = 0$
3. R, G, B in [0, 1]

3.5.4 Use of the LUV model

The LUV model is an approximation of a uniform colour space and is recommended by the CIE for use in colour difference evaluation. Unfortunately, it is still only an approximation. If a better uniform colour space model were to be defined, all the advantages of the LUV model would still hold for the new model since the LUV model's advantages stem from its uniformity, not from the LUV model as such.

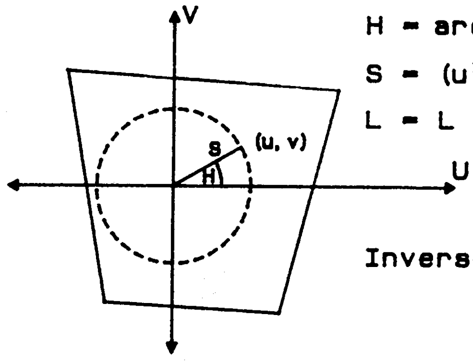
The LUV model is not very suitable for interactive applications. Its conversions are relatively slow (a cubic root must be computed), and it is difficult to acquire a "feeling" for the model, that is, to know where certain colours are located in the colour space.

Another disadvantage is the fact that each component of the LUV model has its own range. If R, G and B range between 0 and 1, L, U and V will range between 0 and 100, –93 and 204, and –132 and 104 respectively.

In order to provide a uniform colour space, the LUV model must be sensitive to environmental factors. This further complicates matters. External factors, such as the phosphors used in the colour display unit and the light source illuminating the display unit, all have an influence on the colours perceived by the human eye.

In spite of the disadvantages, the LUV model can be useful for uniform colour space applications, such as automatic colour assignment (see 4.4). An example is the assignment of colour to a vegetation DTM. Typically, one would want the different vegetation areas to be displayed with colours that are as different as possible. This could be done by spacing the colours as far as possible from one another in LUV space.

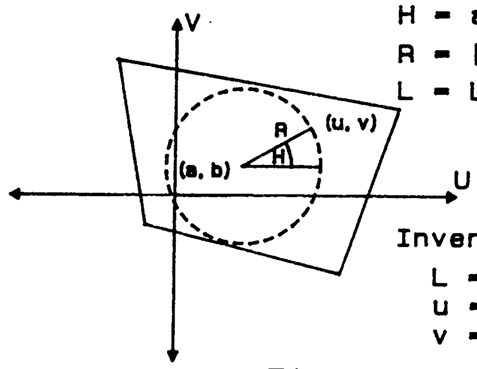
For a digital elevation model (showing the heights of an area) one would want the colours to show the increase in height. This may be done by selecting colours along a line in LUV space. If the intervals between the colours on the line are equal to one another, the height increase will be represented



$$H = \arctan\left(\frac{v}{u}\right)$$
$$S = (u^2 + v^2)^{\frac{1}{2}}$$
$$L = L$$

Inverse:
$$L = L$$
$$u = S\cos(H)$$
$$v = S\sin(H)$$

Fig 15
Definition of H, S and L in terms of the LUV colour model



$$H = \arctan\left(\frac{v-b}{u-a}\right)$$
$$R = [(u-a)^2 + (v-b)^2]^{\frac{1}{2}}$$
$$L = L$$

Inverse:
$$L = L$$
$$u = a + R\cos(H)$$
$$v = b + R\sin(H)$$

Fig 16
Definition of H, R and L with the center of the circle at (a, b)

uniformly. That is, small difference in colour will imply a small difference in height.

In [17] a natural, user-friendly way of specifying LUV colours along a curve is given. For a given L value the U and V components are specified in polar coordinates, called S (saturation) and H (hue) (see figure 15). L, S and H are similar to the HSL model, except that uniformity is preserved for each component. The advantages of this are obvious.

The size of the hue circle could be improved if centres other than (0,0) are used (figure 16). This will increase the colour space utilisation at the cost of saturation changes along the circumference of the hue circle.

If an image has to represent three input functions, the LUV model could be used, together with statistics on the input functions, to provide an "optimal" colour assignment [15]. Refer to 4.4 for more detail on this.

3.6 Choosing the Right Colour Model

The selection of a colour model depends on the intended application. An application where the user interactively specifies or changes colours may be best served by one of the HSI, HSL or CNS models. The CNS model allows the use of natural English names, but has a coarse resolution. The HSI and HSL models allow fine resolution and appear natural to the user. The HSI model corresponds to the artist's use of tint, shade and tone. The conversions needed for the HSI and HSL models are fast, and both models can be used in painting programs. Whether HSI and HSL should be used depends on whether the user wants the fully saturated colours to be at I = 1 (for HSI) or at L = 0.5 (for HSL).

For applications needing a uniform colour space, the LUV model is the obvious choice. It must, however, be remembered that the conversions needed for this model are slower than for the other models.

4. Colour Assignment

In this section it is assumed that the hardware used for colour display consists of a frame buffer and a colour lookup table (colour map or luvo). Each pixel in the frame buffer points to a location in the colour map that contains the colour description of the pixel. In this way the colours may be changed without the picture being changed. The size of the colour map is much less than that of the frame buffer. Typically, the frame buffer may have a size of 512 by 512, giving 262 144 pixels, while the colour map may have only 256 entries. These values will be used in the following discussion.

4.1 Displaying Multi-Image Pictures

The process of displaying digital multi-image pictures on the hardware described above can be seen as consisting of two steps, namely quantisation and colour assignment [18].

Quantisation is the process of assigning representation values to ranges of input values. For example, the 262 144 possible pixel values of the frame buffer must be limited to 256 — the size of the colour map — and all other pixel values must be mapped onto these 256 values. Colour assignment is then simply the assignment of representative colours to the colour map by using one of the colour models.

Typical applications where these steps are needed are: mapping of a video image with R,G,B components onto a k-sized colour map [16]; the false colouring of satellite images; and the combined display of several DTM's, such as DTM's of the soil type, vegetation and slope of an area.

A more formal description of the process of combining and displaying m input images is as follows (see also figure 17).

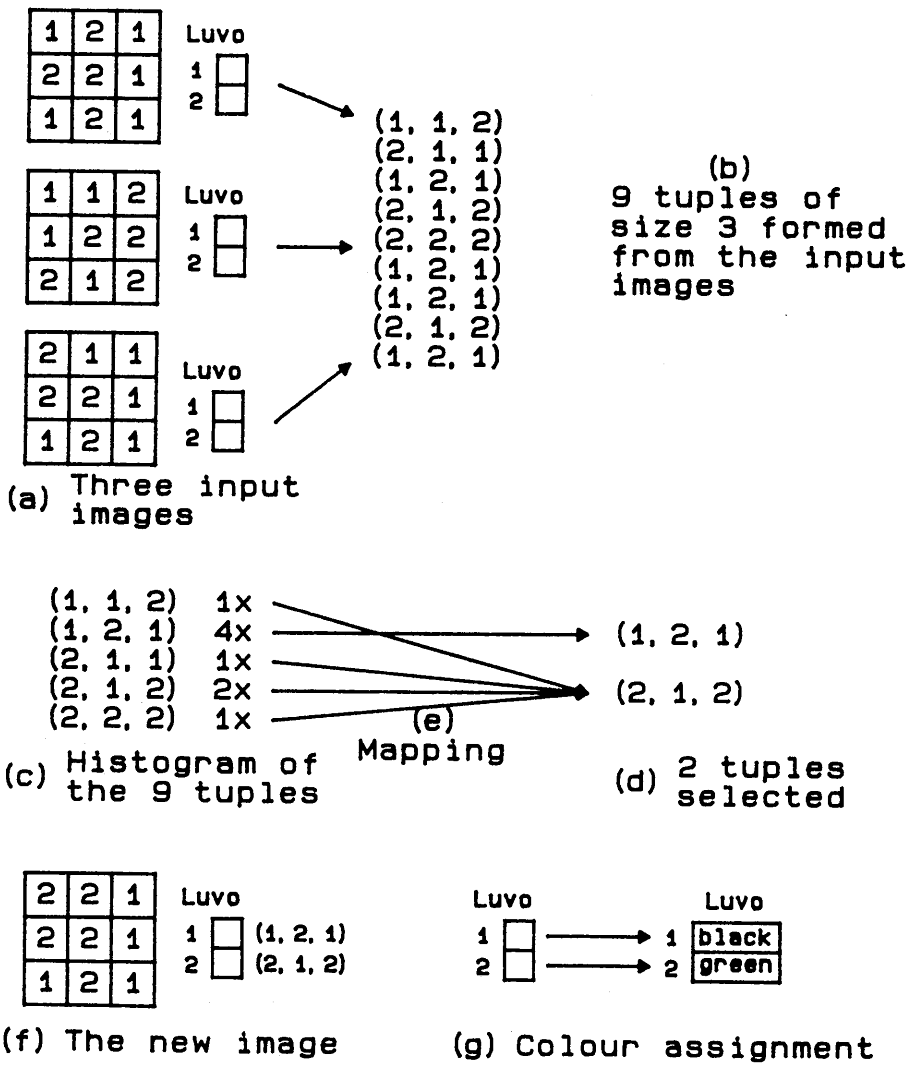


Figure 17 Combining 3 input images (size 3 x 3) into one

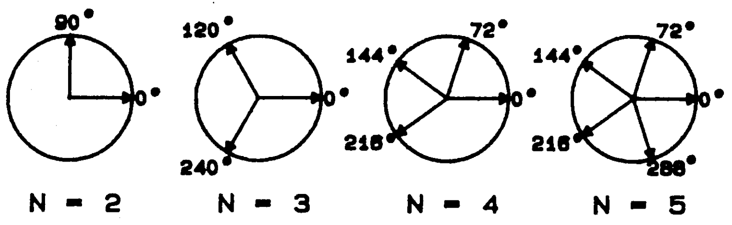


Figure 18 Polar coordinate system for N = 2, 3, 4 and 5

1. Input: Start with n tuples of size m, where a tuple consists of the m input values that have to be represented by one pixel in the final image.
2. Quantisation: Quantise the n tuples to k tuples, where k is the size of the colour map. This is done in three steps (see [18]):
   - (a) Obtain the distribution of the n tuples (a histogram of the tuples).
   - (b) Select k tuples based on the tuple distribution. Typically, select the k tuples that occurred most frequently.
   - (c) Map all the other tuples onto the k tuples selected in (b), and create a new image.
3. Colour assignment: Assign k colours to the k tuples by using the RGB, HSI, HSL or LUV model. Some of the more detailed requirements for this process might be:
   - (a) Use a set of colours with an easily remembered order, showing the order of the tuples.
   - (b) Select colours that are perceived as being as different as possible by the user.
   - (c) Ensure that the colours preserve the distance between the tuples.

4.2 Colour Assignment using the HSI Model

For multi-image pictures the colour of an entry in the colour map might be defined as a function of all the image components. For three or less image components a straightforward assignment of a colour component (hue, saturation or intensity) to an image component may be done. The RGB colour space may also be used in this way, but the use of the HSI colour space is said to give better results. This is evident if there is only one image component because the hue colour component has a large dynamic range. According to [6], about 128 hues can be distinguished by the human eye.

For two image components saturation is usually selected as the second colour component because most displays offer the smallest number of discernable steps along the intensity range. If colours differ only in saturation, we can distinguish from 16 (for yellow) to 23 (for red and magenta) colours. Three image components will use hue, saturation and intensity.

It is also possible to assign one colour component as a function of more than one image component. In this way more than three image components can be displayed at the same time. Some information will be lost because more than one set of image component values will be mapped onto the same colour, but such a multi-image colour picture can still be useful for four or five image components.

The colour assignment for two to five image components may be defined as follows [13]:

$$I = \max\{C_i\}$$

$$S = \frac{\max\{C_i\} - \min\{C_i\}}{\max\{C_i\}}$$

$$H = \arctan\left(\frac{\sum_{i=1}^{N} C_i \sin\theta_i}{\sum_{i=1}^{N} C_i \cos\theta_i}\right)$$

WHERE

N = number of image components ($2 \le N \le 5$)

C_i = value of image component i ($1 \le i \le N$)

θ_i = direction of component image axis in polar coordinate system (see fig. 18)

$$(\theta_1, \theta_2, \ldots \theta_i) = \begin{cases} (0°, 90°) & \text{if } N=2 \\ (0°, 120°, 240°) & \text{if } N=3 \\ (0°, 72°, 144°, 216°) & \text{if } N=4 \\ (0°, 72°, 144°, 216°, 288°) & \text{if } N=5 \end{cases}$$

For N = 3, the choice for the angles of the component axes corresponds to the primary colours. Arrangements where two component axes are directly opposite one another in the polar system are avoided as the two components will cancel one another, since $\sin(180 + k) = -\sin(k)$ and $\cos(180 + k) = -\cos(k)$.

4.3 Colour Assignment using the HSL Model

The colour assignment used for the HSI model may be adjusted for use with the HSL model. However, for the latter model the difference between the S and L colour components may not be as obvious as with the HSI model. If there are two image components that are assigned directly to a colour component, the H and L components may be preferred to the H and S components. This is because L offers a wide range, going from black through the pure colour to white.

If a colour component is assigned as a function of more than one image component, the assignment will be as follows:

$$L = \frac{\max\{C_i\} + \min\{C_i\}}{2}$$

$$S = \begin{cases} \dfrac{\max\{C_i\} - \min\{C_i\}}{\max\{C_i\} + \min\{C_i\}} & \text{if } L \leq 5 \\ \dfrac{\max\{C_i\} - \min\{C_i\}}{2 - \max\{C_i\} - \min\{C_i\}} & \text{if } L > 5 \\ 0 & \text{if } L = 0 \text{ or } L = 1 \end{cases}$$

$$H = \arctan\left(\frac{\sum_{i=1}^{N} C_i \sin\theta_i}{\sum_{i=1}^{N} C_i \cos\theta_i}\right)$$

WHERE

N = number of image components ($2 \leq N \leq 5$)

C_i = value of image component i ($1 \leq i \leq N$)

θ_i = direction of component image axis in polar coordinate system (see fig. 18)

$$(\theta_1, \theta_2, \ldots\theta_i) = \begin{cases} (0°, 90°) & \text{if } N=2 \\ (0°, 120°, 240°) & \text{if } N=3 \\ (0°, 72°, 144°, 216°) & \text{if } N=4 \\ (0°, 72°, 144°, 216°, 288°) & \text{if } N=5 \end{cases}$$

The definitions of S and L differ from the HSI assignment, while the H component is exactly the same.

4.4 Colour Assignment using the LUV Model

In a manner similar to that for the HSI and HSL colour models, colour assignment in LUV space may be done by the assignment of L, U and V to each of the three image components. In the LUV space the axis offering the greater colour variation lies along the U axis (that is, green to red) while the least variation is found along the L axis (black to white). If this colour assignment is used, some problems may arise because even if the individual values of L, U and V are within the specified limits, the point they represent in LUV space may be outside the limits. This is because of the form of the RGB colour cube in LUV space (see figure 14).

If there is only one image component, colours may be assigned by going along a line (or circle, or some similar shape) in LUV space. If the colours are assigned at regular intervals along the line, the colours should appear to be equidistant to the human eye (because of the uniformity of LUV). For two image components a plane (or other geometric figure) in LUV space has to be used.

A method for assigning LUV colours to an image with three components is given in [15]. This method uses statistics on the distribution of the image components' values. A transformation that will fit the component space "optimally" into the LUV colour space is deduced. The formulas used are as follows:

GIVEN:

$$\begin{bmatrix} \bar{L} \\ \bar{U} \\ \bar{V} \end{bmatrix} = \begin{bmatrix} 58.1135 \\ 21.9842 \\ -5.7558 \end{bmatrix}$$ = mean values of regular samples in the displayable LUV space (using D65 for the 1984 observer).

$$\begin{bmatrix} d_{01} \\ d_{02} \\ d_{03} \end{bmatrix} = \begin{bmatrix} 50.5489 \\ 46.6821 \\ 18.6878 \end{bmatrix}$$ = standard deviations of the samples in LUV space in decreasing order.

$$R_0 = \begin{bmatrix} 0.157303 & 0.0857109 & 0.983824 \\ -0.884021 & 0.456285 & 0.101593 \\ 0.440185 & 0.885696 & -0.147548 \end{bmatrix} =$$ matrix such that each column is a normalised eigenvector of the covariance matrix for the displayable LUV space. The eigenvectors are ordered such that their eigenvalues are in decreasing order.

Set of (f_1, f_2, f_3) triples. formed by the three image components f_1, f_2, and f_3.

DESIRED:

The (L, U, V) triple corresponding to (f_1, f_2, f_3), using a data dependent conversion that does not preserve distances.

FORMULA:

As above, calculate the following for the $f_1f_2f_3$ space:

$$\begin{bmatrix} \bar{f_1} \\ \bar{f_2} \\ \bar{f_3} \end{bmatrix}$$ = mean distribution. $$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$ = standard deviation

R = matrix of normalised eigenvectors similar to R_0, but for this space (inverse is R^{-1}).

The conversion is given by:

$$\begin{bmatrix} L \\ U \\ V \end{bmatrix} = R_0 \begin{bmatrix} \frac{d_{01}}{d_1} & 0 & 0 \\ 0 & \frac{d_{02}}{d_2} & 0 \\ 0 & 0 & \frac{d_{03}}{d_3} \end{bmatrix} R^{-1} \begin{bmatrix} f_1 - \bar{f_1} \\ f_2 - \bar{f_2} \\ f_3 - \bar{f_3} \end{bmatrix} + \begin{bmatrix} \bar{L} \\ \bar{U} \\ \bar{V} \end{bmatrix}$$

To preserve distances, replace the $\frac{d_{0i}}{d_i}$ values with their minimum.

5. Guidelines for Effective Colour Use

The following guidelines are based on [2] (see also [19] and [20]) with some remarks added from own experience. The guidelines are listed according to their area of derivation physiological, perceptual, or cognitive.

5.1 Physiological Guidelines

Avoid the simultaneous display of highly saturated, spectrally extreme colours. It is said that extreme colour pairs, such as red and blue or yellow and purple, should be avoided since they cause frequent refocusing and visual fatigue. This strain on the eye is not always felt immediately, but may show itself in the long run. However, desaturation of spectrally extreme colours or the reduction of their intensity will reduce the need for refocusing.

Avoid pure blue for text, thin lines, and small shapes. Our visual system cannot deal adequately with detailed, sharp, short-wavelength stimuli. However, blue does make a good background colour and is perceived clearly out into the periphery of our visual field.

Avoid adjacent colours differing only with regard to the amount of blue. Edges that differ only with regard to the amount of blue will appear indistinct. This is especially visible in the blue-green and yellow-white (where blue is added) regions.

Older viewers need higher levels of brightness to distinguish colours.

Colour changes appearance as ambient light level changes. Displays change colour under different kinds of ambient light — fluorescent, incandescent, or daylight. Appearance also changes as the light level is increased or decreased. On the one hand, a change is due to increased or decreased contrast, and on the other, it is due to a shift in the eye's sensitivity. The background colour of a display has similar effects.

The magnitude of a detectable change in colour varies across the spectrum. Small changes in hue are more detectable in yellow, magenta and cyan-blue than in green and the extreme reds and purples. Small changes in saturation are more difficult to detect in yellow, green and blue than in red-yellow, cyan-blue and blue-magenta. Intensity changes are more visible in green and yellow than in blue.

Difficulty in focusing results from edges created by colour alone. Our visual system depends on a difference in brightness at an edge to effect clear focusing. Therefore, multi-coloured images should be differentiated on the basis of brightness as well as of colour.

Avoid red and green in the periphery of large-scale displays. Owing to the insensitivity of the retinal periphery to red and green, these colours should be avoided in saturated form, especially for small symbols and shapes. Blue and especially yellow are good peripheral colours.

Opponent colours go together well. Red and green or yellow and blue are good combinations for simple displays. The opposite combinations — red with yellow or green with blue — produce poorer images.

For colour-deficient observers, avoid single-colour distinctions.

5.2 Perceptual Guidelines

Not all colours are equally discernible. Perceptually, we need a large change in wavelength to perceive a colour difference in some portions of the spectrum and a small one in other portions. Colour differences are perceived more readily in the yellow-red, cyan-blue and blue-magenta colour regions.

Luminance does not equal brightness. Two colours of equal luminance but different hue will probably appear to be of differing brightness. The deviations are most extreme for colours towards the ends of the spectrum (red, magenta, blue).

Different hues have inherently different saturation levels. Yellow in particular always appears to be less saturated than other hues.

Lightness and brightness are distinguishable on a printed hard copy, but not on a colour display. The nature of a colour display does not allow lightness and brightness to be varied independently (see 2.1 for the definition of these terms).

Not all colours are equally readable or legible. Extreme care should be exercised with text colour relative to background colours. In addition to causing a loss in hue with reduced size, inadequate contrast frequently results when the background and text colours are similar. As a general rule, the darker, spectrally extreme colours such as red, blue magenta, brown, etc., make good backgrounds while the brighter, spectrum-centred, and desaturated hues produce more legible text.

Hues change with intensity and background colour. When grouping elements on the basis of colour, ensure that backgrounds or nearby colours do not change the hue of an element in the group. Limiting the number of colours and ensuring that they are widely separated in the spectrum will reduce confusion.

Avoid the need for colour discrimination in small areas. Hue information is lost in small areas. In general, two adjacent lines of a single-pixel width will merge to produce a mixture of the two. Also, the human visual system produces sharper images with achromatic colours. Thus, for fine detail, it is best to use black, white, and grey, while chromatic colours should be reserved for larger panels or to attract attention.

5.3 Cognitive Guidelines

Do not overuse colour. The best rule is probably to use colour sparingly. The benefits of colour as an attention getter, information grouper, and value assigner are lost if too many colours are used. Cognitive scientists have shown that the human mind experiences great difficulty in maintaining more than five to seven elements simultaneously; so it is best to limit displays to about six clearly discriminable colours when a definite meaning is associated with each colour. In some applications, such as where different shades of the same colour is used, this rule does not apply.

Be aware of the nonlinear colour manipulation in video and hard copy. Video or hard-copy systems cannot match human perception and expectations in all respects.

Group related elements by using a common background colour. Cognitive science has advanced the notion of set and preattentive processing. In this context, you can prepare the user for related events by using a common colour code. A successive set of images can be shown to be related by the use of the same background colour.

Similar colours connote similar meanings. Elements related in some way can convey the relationship by means of the similarity of their hues. The colour range from blue to green is experienced as being more similar than that from red to green. The saturation level can also be used to connote the strength of relationships.

Brightness and saturation draw attention. The brightest and most highly saturated area of a colour display immediately draws the viewer's attention. Yellow and green are good examples of this.

Link the degree of colour change to event magnitude. As an alternative to bar charts or tic marks on amplitude scales, displays can portray magnitude changes with progressive steps of changing colour. A desaturated cyan can be increased in saturation as the graphed elements increase in value. Progressively switching from one hue to another can be used to indicate passing critical levels.

Order colours by their spectral position. To increase the number of colours on a display requires that a meaningful order be imposed on the colours. The most obvious order is that provided by the spectrum with the mnemonic ROY G. BIV (red, orange, yellow, green, blue, indigo, violet). This is the same as the hue range of the HSI and HSL colour models.

Warm and cold colours should indicate action levels. Traditionally, the warm (long wavelength) colours are used to signify action or the requirement of a response. Cool colours, on the other hand, indicate status or background information. Most people also experience warm colours such as red as advancing toward them, hence forcing attention, and cool colours such as blue as receding or drawing away.

Do not mix the decorative use of colour with planned colour cueing. The logical side of our brain looks for meaning in the use of colour and will get tired and frustrated if none could be found [20].

The three-dimensional appearance of objects can be enhanced by the use of variations in the degree of saturation and/or intensity. Foreground colours are commonly rendered in bright colours and the background colours progressively reduced in saturation (or intensity) as the distance from the front increases [19].

6. Conclusion

While these guidelines offer some suggestions, they should certainly not be taken as binding under all circumstances. There are too many variables in colour display, colour copying, human perception, and human interpretation for any hard and fast rules to apply at all times, thus leaving open the way for experimentation.

References

[1] R. Shoup, [1979], Color Table Animation, *Computer Graphics*, **13** (2), 8 - 13.
[2] G.M. Murch, [1985], Colour Graphics Blessing or Ballyhoo?, *Computer Graphics Forum*, **4**, 127 - 135.
[3] G.S. Ellis, [1983], SCRAP Gebruikershandleiding, Special Report SWISK 29, NRIMS, CSIR, South Africa.
[4] P.P. Roets, [1983], IKSP (Ikonas Support Package), Internal Report 1524, NRIMS, CSIR, South Africa.
[5] D. Judd and G. Wyszecki, [1975], *Color in Business, Science and Industry*, John Wiley and Sons, New York.
[6] J.D. Foley and A. van Dam, [1983], *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, USA.
[7] A.R. Smith, [1978], Color Gamut Transform Pairs, *Computer Graphics*, **12** (3), 12 - 19.
[8] G.H. Joblove and D.P. Greenberg, [1978], Color Spaces for Computer Graphics, *Computer Graphics*, **12** (3), 20 - 27.
[9] Status Report of the Graphics Standards Committee, *Computer Graphics*, **13** (3), 1979, III-37 - III-39.
[10] B. Meier, [1985], BUCOLIC : A Program for Teaching Color Theory to Art Students, *IEEE Computer Graphics and Applications*, 57 - 65.
[11] A.R. Smith, [1979], Tint Fill, *Computer Graphics*, **13** (2), 276 - 283.
[12] K.R. Sloan and C.M. Brown, [1978], Color Map Techniques, *Computer Graphics and Image Processing*, **10**, 297 - 317.
[13] W. Fink, [1976], Image Transformations Facilitating Visual Interpretation, Goddard Space Flight Center, Greenbelt, Maryland, June.
[14] T. Berk, L. Brownston, and A. Kaufman, [1982], A New Color-Naming System for Graphics Languages, *IEEE Computer Graphics and Applications*, 37 - 44.
[15] J. Tajima, [1983], Uniform Color Scale Applications to Computer Graphics, *Computer Vision, Graphics, and Image Processing*, **22** (1), 305 - 325.
[16] G.W. Meyer and D.P. Greenberg, [1980], Perceptual color spaces for computer graphics, *Computer Graphics*, **14** (3), 254 - 261.
[17] P.K. Robertson and J.F.O. O'Callaghan, [1986], The Generation of Color Sequences for Univariate and Bivariate Mapping, *IEEE Computer Graphics and Applications*, **6** (2), 24 - 32.

[18] P. Heckbert, [1982], Colour Image Quantization for Frame Buffer Display, *Computer Graphics*, **16** (3), 297 - 305.

[19] J.R. Truckenbord, [1981], Effective use of color in computer graphics, *Computer Graphics*, **15** (3), 83 - 90.

[20] H.J. Ehlers, [1985], The use of colour to help visualize information, *Computers and Graphics*, **9** (2), 171 - 176.

BOOK REVIEW

Effective Computer Applications

by Peter Pirow, 1986, Woodacres Publishers, 230 pages, ISBN 0 620 09323 4

Dr. Peter Pirow has made a most original and sorely needed longitudinal, case-history research contribution to the cost-effectiveness of business information systems in particular, and to the study of the social use of computers in general. *Effective Computer Applications* reflect a massive research effort spanning a 28 year period, incorporating a selection of 857 case studies in South Africa over an astonishing variety of applications and organisations. The extensive case study empirical data base was subjected to diversified hypothesis testing to determine various socio-economic and technical facets of computer system performance effectiveness. This carefully crafted work, including a wide-ranging general history of information and computers, and a special history of computer developments and applications in South Africa, is a remarkable intellectual tour de force and an exemplary, interdisciplinary scholarly task.

Voluminous descriptive statistics are presented, highlighting case history population parameters and characteristics, and quantitative measures of computer system success and failure. Numerous statistical tests are made of leading hypotheses and organisational performance measures culled from the scientific literature, with additional hypotheses and models from Dr. Pirow's research. These statistics include parametric, non-parametric, correlational and multivariate techniques appropriate for the empirical samples. The longitudinal quantitative analyses over almost a three-decade time period is clearly a major contribution to the world literature in business information systems. The case history and quantitative methodology is of great value, and I felt that it should have been explicated more fully for applied scientific practitioners, particularly in connection with empirical reliability and validity of the various measures of performance effectiveness.

The book concludes with a thoughtful analysis of the complex reasons why so many information systems fail and others succeed in various organisational contexts. Dr. Pirow particularly singles out the massive impact of computer illiteracy on system failures, and provides broad educational recommendations for achieving "computeracy" at the national level. He ends on a fitting sober, scientific note that the proverbial wisdom appearing in newspapers, magazines, and articles, based on "expert" opinion as to what works and what does not work in computer systems, "can only be tested by means of a study of a substantial number of applications", as he has done in this monumental work.

Hal Sackman, *School of Business, California State University*

Using NLC-Grammars to Formalise the Take/Grant and Send/Receive Security Models

D P de Villiers and S H von Solms
Department of Computer Science, Rand Afrikaans University, P O Box 524, Johannesburg, 2000

Abstract

In this paper the Send/Receive and Take/Grant logical security models are formalised using results from formal language theory. By using the graph rewriting facilities of NLC-grammars, and by extending these facilities to take different types of context conditions into account, the actions within the Send/Receive and Take/Grant models are simulated.

Keywords: *Graph grammars, Node-Label-Controlled graph grammars, Grammatical protection systems, Take/Grant models, Send/Receive models*

Received March 1987, Accepted July 1987

1. Introduction

The goal of this study is the definition of a context-dependent logical security model. This model will use the current state of the protected environment to determine whether a new access relation between two entities in the environment may be established, or whether such an existing relationship may be altered. Furthermore, when new entities are introduced into the environment, the current protection state will also be used in determining the relationship between the new and existing entities. A motivation for this is to formalise the definition of the access rights of new users in a protected system. Instead of having to manually determine the access rights of each user, the protection system will define the access rights in view of the current system context. This study will provide the basis for a logical formulation of such a model.

Firstly, current models of access control were studied, to find a suitable framework for development of the envisaged model. It was found that the T/G (Take/Grant) and S/R (Send/Receive) models [4,3] showed promise, for the following reasons

- it is possible to ignore the subject/object distinction.
- the graph-theoretic approach and related graph-rewriting rules pointed to the possible incorporation of a suitable graph grammar, that would make a large body of proven theory available for use in the study.
- the T/G and S/R models are enjoying considerable attention. New results, that appear quite frequently, may lead to new ideas in this study.

We will now give short summaries, first of the T/G model (from [4]), and then of the S/R model (from [3]).

The object of the T/G model is to model a protection system. This is done by using a two-coloured, directed graph, wherein subjects and objects are represented by different coloured nodes of the graph, and where the capability of node a to access node b is indicated by an edge from a to b. The set α of rights that a can exercise in accessing b (such as read, write) is denoted by labelling the edge from a to b with α, as follows:

$$\begin{tikzcd} a \arrow[r, dash, "\alpha"] & b \end{tikzcd}$$

The set α of rights is a subset of the fixed and finite set of rights *R* = {*read, write, append, execute* (programs), *take, grant*}. The *read, write, append* and *execute* rights are called the inert rights, while the *take* and *grant* rights may be seen as "active" rights, in that they allow the modification of the protection graph structure.

The T/G model also defines certain graph rewriting rules, two of which correspond to the *take* and *grant* rights. Informally, if node a has the *take* right to node b, and the *grant* right to node c, node a may take any subset of the rights of node b for itself, and give (grant) any subset of its own rights to node c.

There are two more rewriting rules in this formulation of the T/G model, namely the create and remove rules. The create rule adds a new node to the graph, with a labelled edge from the node that initiated the creation operation to the newly created node. The remove rule alters the labelling of an edge of the graph (i.e. an alteration of the access rights of the start node of the edge) by removing a subset of rights from the set of rights denoted by the edge label.

The T/G operations will later be more formally defined.

Like the Take/Grant (T/G) model, the purpose of the Send/Receive model is the modelling of the transfer of access rights between subjects in a protected system.

In the S/R model for protection, several shortcomings of the Take/Grant model are addressed, including *selectivity*, *locality/modularity*, and *unidirectionality* of flow of rights (see [3] for more detail).

We shall now give a brief definition of the S/R model, from [3]. In this model, a protected system consists of typed objects. With each object **x** we associate a set of (*attribute*, *value*) tuples, describing the attributes of the object, as well as a (possibly empty) set of (*object*, *access right set*) tuples, called *tickets*, which denotes the fact that **x** can access the given *object* with any of the rights in the set *access right set*. The *access right set* is a subset of a fixed and finite set of rights R = {*read*, *write*, *append*, *execute* (programs), *send*, *receive*}. Finally, an object also possesses a (potentially empty) set of *rule rights*, called *activators/rules*. An activator takes the following (simplified) form:

CAN A(a_1, ..., a_k) IF Q(a_1, ..., a_k).

This means that the holder of the activator may perform action A on the arguments (logical entities/objects) a_1, ..., a_k, provided that the predicate Q is satisfied.

In the rest of this paper, we shall refer to activators as rules.

Three basic rules are defined in the S/R model.

The first rule, called the CAN-SEND rule, facilitates the control of movement of rights out of an object's domain, enabling the holder thereof to send rights to another object. The rule takes the following form:

CAN SEND p:P TO s:S IF Q(p, s).

p and s are free variables and are only present if they are used within the predicate Q. P and S are tuples, called *templates*, of the form (T, R), where (for P) T represents the type of the object p, and R represents the set of rights of p that may be sent to the object s. These templates (called *patterns* in [3]) are matched by a ticket (*object*, *access right set*) when the *object* is of type T and the *access right set* is contained in the set R. S will always have the form (T, *send*). This type of rule may be activated only if its holder possesses tickets that match P and S, and Q(p, s) is satisfied.

The second rule enforces control over the movement of rights into an object's domain, enabling its holder to receive rights from another object. The rule takes the following form:

CAN RECEIVE p:(T, R) FROM s:(T, *receive*) IF Q(p, s).

This rule may be activated only if its holder possesses the *receive* right to the object s from which it wants to receive a ticket of the form (T, R), and if the predicate Q is satisfied.

Finally, there is a CREATE operation for the creation of new objects. It is assumed that if an object **x** wants to create a new object **y**, **x** receives the tickets (**y**, *send*), (**y**, *receive*), and **y** receives the tickets (**x**, *send*), (**x**, *receive*), so that full bidirectional transfer of rights is possible. The form of the CAN-CREATE rule is not defined in [3], and for the purposes of this article we define it as follows:

CAN CREATE p:(T, $\mathcal{R}$) WHEN (T, *create*) IF Q(p, s).

This means that the holder of the rule can create an object p of type T subject to the condition that the predicate Q is satisfied. The set $\mathcal{R}$ is a subset of the set of rights possessed by the holder of the CAN-CREATE rule.

Finally, in order for a ticket (t, r) to be transferred from an object **x** to an object **y**, the following must be true:

- **x** must possess the tickets (t, r) and (**y**, *send*), as well as an appropriate CAN-SEND rule, that is satisfied by these tickets.
- **y** must possess the ticket (**x**, *receive*), as well as an appropriate CAN-RECEIVE rule, that is satisfied by the tickets (**x**, *receive*) and (t, r).

As was said earlier, the incorporation of a formal graph grammar is indicated by the graph-theoretic approach and rewriting rules of the T/G model. Of existing graph grammars, the so-called NLC-(Node-Label Controlled) grammars [1] seem to be the most applicable. An important motivation is that the grammars were extended by von Solms [6,7,8] to include permitting and forbidding node contexts in their productions. As context dependency is a fundamental aspect of the grammar to be used, the grammar was found to be very acceptable in this regard.

We shall now proceed with the necessary definitions for simulating the T/G model, and then discuss some of the aspects of the resulting model, which we will call the CSM (Context-dependent Security Model). This model is fully described in [10].

2. Extending NLC-grammars to Formalise the T/G Model

For the definition of a node-labelled undirected graph, an NLC grammar and related concepts, see [1].

Definition 2.1 (from [1])

A NLC-grammar with node context is a system

G = (Σ, Δ, P, C, Z) where

Σ is a nonempty finite set of labels, called the total alphabet

Δ is a nonempty finite subset of Σ, called the terminal alphabet

P is a finite set of productions, of the form (d, D, (Σ_1 ; Σ_2)) with

$d \in \Sigma$

D a graph over Σ

$\Sigma_1, \Sigma_2 \subseteq \Sigma$

$\Sigma_1 \cap \Sigma_2 = \varnothing$, where

Σ_1 is called the permitting- and Σ_2 the forbidding (node) context

C is a subset of $\Sigma \times \Sigma$, called the connection relation

Z is a graph over Σ, called the axiom.

Definition 2.2

Let R be a finite, nonempty set of symbols, called the set of rights. Let H be a graph over Σ, and let e =

$$\begin{tikzcd} l_1 \arrow[r, dash] & l_2 \end{tikzcd}$$

be a subgraph of H, consisting of any two connected nodes of H and the edge between them. We associate with the edge of e an $\alpha \subseteq R$, called the edge classification. We denote this by a tuple (l_1, l_2, α), called an edge context. The edge context is denoted graphically by adding the labels l_1,α to the edge, as follows:

$$\begin{tikzcd} l_1 \arrow[r, dash, "l_1{,}\alpha"] & l_2 \end{tikzcd}$$

The edge context (l_2, l_1, β) is written

$$\begin{tikzcd} l_2 \arrow[r, dash, "l_2{,}\beta"] & l_1 \end{tikzcd}$$

We say that l_1 owns the set α of rights with respect to l_2. The "owner" of the rights is therefore indicated by being the first element of the tuple.

The set Λ_H of all possible edge contexts of a graph H is defined as

$\Lambda_H = \{(l_i, l_j, \alpha) \mid$ $\begin{tikzcd} l_i \arrow[r, dash] & l_j \end{tikzcd}$ is an edge of H, $i \neq j, \alpha \subseteq R\}$,

where R is the set of rights associated with the grammar that generated H.

For a grammar G, Λ_G is defined as

$\Lambda_G = \{(l_i, l_j, \alpha) \mid l_i, l_j, \in \Sigma, l_i \neq l_j, \alpha \subseteq R\}$,

where R is the set of rights associated with G.

Definition 2.3

A NLC-grammar with node- and edge context is a system

G = $(\Sigma, \Delta, P, C, Z, R)$ where

Σ is a nonempty finite set of labels, called the total alphabet

Δ is a nonempty finite subset of Σ, called the terminal alphabet

P is a finite set of productions, of the form

(d, D, $(\Sigma_1 ; \Sigma_2)$, $(\Lambda_1 ; \Lambda_2)$) with

> $d \in \Sigma$
>
> D a graph over Σ, where with every edge of D we associate an edge classification $\alpha \in R$, applicable to one of the nodes of the edge.
>
> $\Sigma_1,\Sigma_2 \subseteq \Sigma$
>
> $\Sigma_1 \cap \Sigma_2 = \varnothing$
>
> Σ_1 is called the permitting and Σ_2 the forbidding node context
>
> $\Lambda_1,\Lambda_2 \subseteq \Lambda_G$. Λ_1 and Λ_2 is called the permitting and forbidding edge contexts, respectively. Also, $\Lambda_1 \cap \Lambda_2 = \varnothing$

C is a subset of $\Sigma \times \Sigma$, called the connection relation

Z is a graph over Σ, called the axiom

R is a finite, nonempty set of symbols, called the set of rights.

The meaning of the permitting and forbidding node/edge contexts in definition 2.3 is as follows: A production may be applied if

- all of the nodes specified in the permitting node context Σ_1 appear somewhere in the current graph
- none of the nodes specified in the forbidding node context Σ_2 appears anywhere in the current graph
- for every edge context (l_1, l_2, α) that appears in the set Λ_1 of permitting edge contexts, there exists an edge somewhere in the current graph between two nodes labelled l_1 and l_2, having an edge classification α.
- for every edge context (l_1, l_2, β) that appears in the set Λ_2 of forbidding edge contexts, there does not exist an edge anywhere in the current graph between two nodes labelled l_3 and l_4, having an edge classification β.

Three types of productions can be identified:

1. Normal productions
2. Edge generation productions
3. Edge removal productions.

A direct derivation step is performed by applying a normal production. Let H be a graph over Σ, with $v \in V_H$, and $\Psi_H(v) = l$. Choose a production $(l, D, (\Sigma_1 ; \Sigma_2), (\Lambda_1 ; \Lambda_2)) \in P_H$, the contexts of which are all satisfied. Apply the production as follows:

(i) Remove v and all edges incident to v from H, resulting in the graph H\v.

(ii) Replace v with an isomorphic copy $\bar{D}$ of D.

(iii) Establish edges between H\v and $\bar{D}$ in the following manner:

> Let $x \in V_{\bar{D}}$, $y \in V_{H\backslash v}$, with $\Psi_{\bar{D}}(x) = l_1$ and $\Psi_H(y) = l_2$.
>
> Create an edge between x and y iff there was an edge between y and v in H and $(l_1, l_2) \in C$.

In order to prevent unauthorised access from taking place, connection of nodes in H\v to nodes in D may be forced to be done explicitly. In such a situation, step (iii) in the application of a normal production, as defined above, would be replaced by

(iii) Establish edges between H\v and v in the following manner:

> Let $x \in V_{H\backslash v}$.
>
> Create an edge between x and v iff there was an edge between x and v in H.

The set of connection relations, C, would therefore not be used at this stage in the determination of connections between nodes in H\v and nodes in D. After (the modified) step (iii) has been executed, any node in $H\backslash v$ wishing to gain access to a node in D would explicitly initiate an edge creation production to create an edge to the node in D. In this way all access to new nodes can be simply but rigorously controlled.

The different types of productions will now be discussed in greater detail.

1. A normal production

is of the form

$(l, D, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$

where D is an arbitrary graph over the set Σ. Let $\bar{D}$ be the isomorphic copy of D that is to replace v; then $V_{\bar{D}} \cap V_{H\backslash v} = \varnothing$. This type of production is denoted by P_n.

This type of production is used where at least one new node is to be introduced into the graph (created). The graph D may consist of one or more nodes, and may contain isolated (unconnected) nodes. A normal production cannot establish edges between existing nodes in the graph; the edge generation production is to be used in such a case.

2. An edge generation production

is of the form

$$\begin{tikzcd}(l_1, l_1 \arrow[r, dash, "{l_1,\alpha}"] & l_2, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))\end{tikzcd}$$

$$\begin{tikzcd}\text{with } l_1 \arrow[r, dash, "{l_2,\beta}"] & l_2 \in \Lambda_2, \forall\ \beta \in R, l_2 \in \Sigma_1, \text{ and}\end{tikzcd}$$

is denoted by P_g.

It is assumed that identical labels in the above production refer to the same nodes.

The function of this type of production is the generation of an edge between two existing nodes labelled l_1 and l_2. This does not exclude the possibility that an edge between the nodes l_1 and l_2 already exists, although we shall see later that it is possible to establish a precondition that no edge may exist between two nodes at the time that a new edge is added between the two nodes.

The creation of the edge is initiated by the node labelled l_1 in order to gain access to the node labelled l_2. The owner of the edge (l_1) gives the *take*, *grant*, as well as all other "inert" (*read*, *write*, *append*, *execute*) rights to the edge classification.

The edge generation production can be graphically illustrated as follows:

$$\begin{tikzcd}l_1 & l_2 \arrow[r, phantom, "\Rightarrow"] & l_1 \arrow[r, dash, "{l_1,\alpha}"] & l_2\end{tikzcd}$$

where the nodes labelled l_1 and l_2 are existing nodes in an arbitrary graph.

3. An edge removal production

is of the form

$$\begin{tikzcd}(l_1, l_1 \arrow[r, dash, dashed, "{l_1,\varnothing}"] & l_2, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))\end{tikzcd}$$

with $(l_1,l_2,\beta) \in \Lambda_1$, $\beta \in R$, and is denoted by P_r.

It is assumed that identical labels in the above production refer to the same nodes.

The function of the edge removal production is to remove an existing edge from the graph, in order to indicate that no relation between the two nodes exists anymore.

The edge removal production can be graphically illustrated as follows:

$$\begin{tikzcd}l_1 \arrow[r, dash, "{l_1,\beta}"] & l_2 \arrow[r, phantom, "\Rightarrow"] & l_1 \arrow[r, dash, dashed, "{l_1,\varnothing}"] & l_2\end{tikzcd}$$

where the nodes labelled l_1 and l_2 are existing and connected nodes in an arbitrary graph.

A fourth production type may also be introduced.

4. Edge classification update production

This type of production is of the form

$$\begin{tikzcd}(l_1, l_1 \arrow[r, dash, "{l_1,\beta}"] & l_2, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2)) \text{ with}\end{tikzcd}$$

$(l_1,l_2,\gamma) \in \Lambda_1$, and is denoted by P_u.

It is assumed that identical labels in the above production refer to the same nodes.

The function of this type of production is to reflect a change in the relationship between two nodes by altering the edge classification, while leaving the nodes and the edge between them intact. When the edge classification becomes empty, it is indicated that no relationship between the two nodes currently exists, although the edge between the nodes is still shown.

The edge classification update production can be graphically illustrated as follows:

$$\begin{tikzcd}l_1 \arrow[r, dash, "{l_1,\gamma}"] & l_2 \arrow[r, phantom, "\Rightarrow"] & l_1 \arrow[r, dash, "{l_1,\beta}"] & l_2\end{tikzcd}$$

where the nodes labelled l_1 and l_2 are existing (connected) nodes in an arbitrary graph.

Another way to handle the removal of edges is to use the edge classification update production to, modify the classification of an edge to be the empty set. In this case, an empty edge classification will denote the situation in which no relation is defined between two nodes, although an edge is shown between them.

Although it may seem that the edge removal production, in contrast to the edge classification update production, would make it possible to distinguish between an isolated node and a node that is connected but with no defined access relations, there is no practical difference between these two situations.

Another remark that is relevant to productions, is that in the NLC-grammars, the sets $V_{H\backslash v}$ and $V_{\bar{D}}$

must be disjoint, i.e. the only node of the graph (to which the production will be applied) that may appear in the right-hand side of the production is the node that appears on the left-hand side of the production, i.e. the node to be replaced. This condition is removed from the CSM, so that the necessary T/G operations may be naturally modelled. It is possible, however, to replace any production that does not obey this rule with a set of productions that do. For more detail, see [8].

Lastly, the following note can be made about the labels of nodes. It is assumed that every label denotes a class of node, e.g. the class of all students. In order to be able to refer in a production to a specific node, a production must first be executed that assigns a unique label to the node. Identification of the node that should receive the unique label can be done by using the node and edge contexts. When the operations necessitating the unique identification of the node have been carried out, the original label of the node may be restored.

3. Modelling of the T/G operations in the modified NLC-grammar

The basic T/G operations will now be formalised in the modified NLC-grammar.

There are four basic operations in the T/G model, namely the Take, Grant, Create, and Remove operations. They are defined as follows: [3]

Let H be graph, with x, y, and z $\in V_H$, $\Psi_H(x) = l_1$, $\Psi_H(y) = l_2$, and $\Psi_H(z) = l_3$.

Let $R = \{t, g, c, d, r, w, a, e\}$ with

t = "*take*" right

g = "*grant*" right

c = "*create*" right

d = "*remove*" ("delete") right

r = "*read*" right

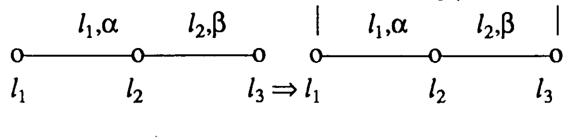
w = "*write*" right

a = "*append*" right

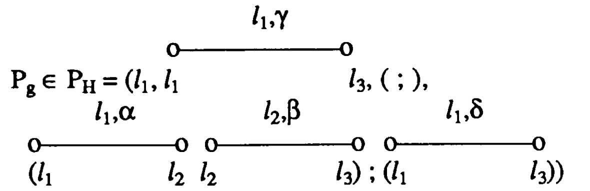
e = "*execute*" right

1. Take

Let there be an edge between x and y with edge classification α with $t \in \alpha$, as well as an edge between y and z with an edge classification β with $\gamma \subseteq \beta$. An application of the Take rule establishes a new edge between x and z with classification γ. The rule can be read as "x takes (γ to z) from y". The graphical representation of this rule is as follows:



This operation is modelled in the CSM by means of the following production:



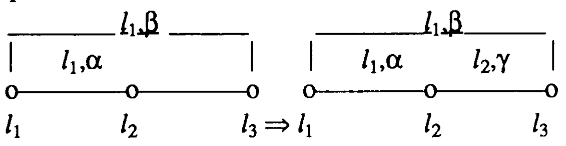
$\forall\ \delta \subseteq R, t \in \alpha$, using the graphical representation of a production, and

$P_g = (l_1, (l_1, l_3, \gamma), (\ ;\ ), (l_1, l_2, \alpha), (l_2, l_3, \beta)\ ;\ (l_1, l_3, \delta))\ \forall\ \delta \subseteq R, t \in \alpha$

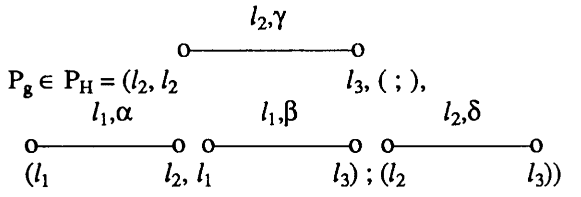
using the "edge context" representation of a production. The graphical representation will be used in the rest of this paper, because it is more legible.

2. Grant

Let there be an edge between x and y with edge classification α, with $g \in \alpha$, as well as an edge between x and z with edge classifcation β, with that $\gamma \subseteq \beta$. The Grant rule defines a new edge between y and z with classification γ. This operation can be read as "x grants (γ to z) to y". The graphical representation is as follows:



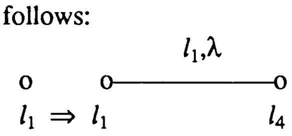
The following CSM production models this operation:



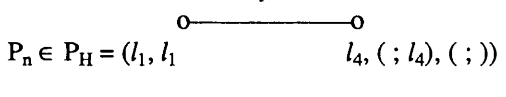
$\forall\ \delta \subseteq R, g \in \alpha, \gamma \subseteq \beta$.

3. Create

Let there be a node $x \in V_H$ in the current graph. The Create rule adds a new node n, with $\Psi_H(n) = l_4$, to the graph, and creates a new edge with classification λ between x and n. This can be read as "x creates (λ to n)". The graphical representation of this rule is as follows:



The following CSM production models this operation:



4. Remove

Let there be an edge between x and n with an edge classification α, with $\beta \subseteq \alpha$. The Remove operation removes the set β of rights from α. This rule may be read as "x removes (β to n)". This operation can be handled by the CSM edge classification update production. The graphical representation is as follows:

$$\begin{tikzcd} l_1 \arrow[r, dash, "{l_1,\alpha}"] & l_4 \arrow[r, phantom, "\Rightarrow"] & l_1 \arrow[r, dash, "{l_1,\alpha-\beta}"] & l_4 \end{tikzcd}$$

The following CSM production models this operation:

$$P_u \in P_H = (l_1, \begin{tikzcd} l_1 \arrow[r, dash, "{l_1,\alpha-\beta}"] & l_4 \end{tikzcd}, (;), (\begin{tikzcd} l_1 \arrow[r, dash, "{l_1,\alpha}"] & l_4 \end{tikzcd};))$$

A similar operation can be used for the addition of rights to an existing edge classification.

The basic T/G operations have now been defined. Note that the node- and edge contexts in the above operations may contain additional nodes and/or edge contexts in real situations, in order to model other constraints dictated by such a situation.

We now proceed to extend the CSM to model the S/R situation. This model is fully described in [11].

4. Extending NLC-grammars to Formalise the S/R Model

Definition 4.1

Let R be a finite, nonempty set of symbols, called the *set of rights*. Let H be a graph over Σ, and let

$$e = \begin{tikzcd} l_1 \arrow[r, dash] & l_2 \end{tikzcd}$$

be a subgraph of H, consisting of any two connected nodes of H and the edge between them. We associate with the edge of e an α, a β and a $\chi \subseteq R$. α, β and χ together constitute the *classification* of the edge.

We denote this by a tuple $(l_1, l_2, \alpha, \beta, \chi)$, called an *edge context*. The edge context is denoted graphically by adding the labels l_1,α,β,χ to the edge, as follows:

$$\begin{tikzcd} l_1 \arrow[r, dash, "{l_1,\alpha,\beta,\chi}"] & l_2. \end{tikzcd}$$

We say that l_1 owns the set α of rights with respect to l_2 and the set β serves the function of *qualifying* the rights in α. That is, if there are any rights within α that operate on other rights, then the rights that are operated upon is given in β and χ respectively, as will be shown later. The "owner" of the rights is indicated by being the first element of the tuple.

The set Λ_H of all possible edge contexts of a graph H, and the set Λ_G of all possible edge contexts of all graphs generated by a grammar G, is defined in the sane manner as in the CSM.

Before we define the grammar that will be used to formalise the S/R model (we shall call it the ECSM - Extended Context-dependent Security Model), a few other preliminary definitions are necessary. We associate with each label l in the protection graph a set of attributes, denoted by $l\{A_1, \ldots, A_k\}$. The value of the attribute A_j is denoted by a_j. The set of all possible values of an attribute a_j is denoted by $\mathcal{D}(A_j)$. We define the attribute set $\mathcal{A}$ as the set of all possible attributes in the system. The power set of $\mathcal{A}$ denoted by $\mathcal{P}(\mathcal{A})$.

Definition 4.2

An *attributed label l* is a label that has associated with it a set of attributes, denoted by

$l\{A_1, \ldots, A_k\}$,

such that $A_1, \ldots, A_k \in \mathcal{A}, l \in \Sigma$, where $\mathcal{A}$ is a set of attributes, and Σ is a nonempty, finite set of node labels.

Definition 4.3

A NLC-grammar with node- and edge context is a system

$G = (\Sigma, \Delta, P, C, Z, R, \mathcal{A})$ where

Σ is nonempty finite set of attributed labels, called the total alphabet

Δ is a nonempty finite subset of Σ, called the terminal alphabet

P is a finite set of productions, of the form

$(d, D, (\Sigma 1 ; \Sigma_2), (\Lambda_1 ; \Lambda_2))$

$d \in \Sigma$

D a graph over Σ, where with every edge D we associate an edge classification α, β, χ with α, $\beta, \chi \subseteq R$, applicable to one of the nodes of the edge;

$\Sigma_1, \Sigma_2 \subseteq \Sigma$

$\Sigma_1 \cap \Sigma_2 = \varnothing$

Σ_1 is called the permitting and Σ_2 the forbidding node context

$\Lambda_1, \Lambda_2 \subseteq \Lambda_G$. Λ_1 and Λ_2 are called the permitting and forbidding edge contexts, respectively. Also, $\Lambda_1 \cap \Lambda_2 = \varnothing$.

C is a subset of $\Sigma \times \Sigma$, called the connection relation

Z is a graph over Σ, called the axiom

R is a finite, nonempty set of symbols, called the set of rights

$\mathcal{A}$ is a finite, nonempty set of attributes.

Definition 4.4

Let $x, y \in V_H$, where H is a graph over Σ, with $\Psi_H(x) = l_1$ and $\Psi_H(y) = l_2$. We say that x *matches* y if

i) $l_1 = l_2$

ii) The value a_j of every attribute A_j of l_1 is the same as the value of the corresponding attribute of l_2.

The meaning of the permitting and forbidding node/edge contexts in definition 4.4 is as follows. A production may be applied if

- all of the nodes specified in the permitting node context Σ_1 are matched by nodes in the current graph
- none of the nodes specified in the forbidding node context Σ_2 is matched by any of the nodes in the current graph

- for every edge context $(l_1, l_2, \alpha, \beta, \chi)$ that appears in the set Λ_1 of permitting edge contexts, there exists an edge somewhere in the current graph between two nodes labelled l_1 and l_2 having an edge classification α, β, χ.

- for every edge context $(l_3, l_4, \varepsilon, \phi, \gamma)$ that appears in the set Λ_2 of forbidding edge contexts, there does not exist an edge anywhere in the current graph between two nodes labelled l_3 and l_4, having an edge classification $\varepsilon, \phi, \gamma$.

As in the CSM, four types of productions can be identified:

1. Normal productions
2. Edge generation productions
3. Edge removal productions
4. Edge classification update productions.

A direct derivation step is performed by following the same steps as in the CSM.

The different types of productions will now be discussed in greater detail.

1. A normal production
is of the form

$(l, D, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2))$

where D is an arbitrary graph over the set Σ. Let $\bar{D}$ be the isomorphic copy of D that is to replace v, with $V_{\bar{D}} \cap V_{H\backslash v} = \varnothing$. This type of production is denoted by P_n.

This type of production is used where at least one new node is to be introduced into the graph (created). The graph D may consist of one or more nodes, and may contain isolated (unconnected) nodes. A normal production cannot establish edges between existing nodes in the graph; the edge generation production is to be used in such a case.

2. An edge generation production
is of the form

$$\Big(l_1, \overset{l_1,\alpha,\beta,\chi}{\underset{l_1}{\circ}\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\underset{l_2}{\circ}}, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2)\Big)$$

with $\overset{l_1,\varepsilon,\phi,\gamma}{\underset{l_1}{\circ}\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\underset{l_2}{\circ}} \in \Lambda_2$, $\forall\ \varepsilon,\phi,\gamma \in R$, $l_2 \in \Sigma_1$, and is denoted by P_g.

It is assumed that identical labels in the above production refer to the sane nodes.

The function of this type of production is the generation of an edge between two existing nodes labelled l_1 and l_2. The creation of the edge is initiated by the node labelled l_1 in order to gain access to the node labelled l_2. The owner of the edge (l_1) gives the *send* and *receive*, as well as all other "inert" (*read, write, append, execute*) rights to the edge classification.

The edge generation production can be graphically illustrated as follows:

$$\underset{l_1}{\circ} \quad \underset{l_2}{\circ} \quad \Rightarrow \quad \overset{l_1,\alpha,\beta,\chi}{\underset{l_1}{\circ}\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\underset{l_2}{\circ}}$$

where the nodes labelled l_1 and l_2 are existing nodes in an arbitrary graph.

3. An edge removal production
is of the form

$$\Big(l_1, \overset{l_1,\varnothing,\varnothing,\varnothing}{\underset{l_1}{\circ}\ -\ -\ -\ -\ \underset{l_2}{\circ}}, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2)\Big)$$

with $(l_1,l_2,\alpha,\beta,\chi) \in \Lambda_1$, $\alpha,\beta,\chi \in R$, and is denoted by P_r.

It is assumed that identical labels in the above production refer to the same nodes.

The function of the edge removal production is to remove an existing edge from the graph, in order to indicate that no relation between the two nodes exists anymore.

The edge removal production can be graphically illustrated as follows:

$$\overset{l_1,\alpha,\beta,\chi}{\underset{l_1}{\circ}\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\underset{l_2}{\circ}} \quad \Rightarrow \quad \overset{l_1,\varnothing,\varnothing,\varnothing}{\underset{l_1}{\circ}\ -\ -\ -\ -\ \underset{l_2}{\circ}}$$

where the nodes labelled l_1 and l_2 are existing and connected nodes in an arbitrary graph.

4. Edge classification update production
This type of production is of the form

$$\Big(l_1, \overset{l_1,\alpha,\beta,\chi}{\underset{l_1}{\circ}\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\underset{l_2}{\circ}}, (\Sigma_1 ; \Sigma_2) , (\Lambda_1 ; \Lambda_2)\Big)$$ with

$(l_1,l_2,\varepsilon,\phi,\gamma) \in \Lambda_1$, while at least one of the following statements is true:

i) $\alpha \neq \varepsilon$

ii) $\beta \neq \phi$

iii) $\chi \neq \gamma$

This type of production is denoted by P_u.

It is assumed that identical labels in the above production refer to the sane nodes.

The function of this type of production is to reflect a change in the relationship between two nodes by altering the edge classification, while leaving the nodes and the edge between then intact. When all the sets in the edge classification becomes empty, it is indicated that no relationship between the two nodes currently exists, although the edge between the nodes is still shown.

The edge classification update production can be graphically illustrated as follows:

$$\overset{l_1,\varepsilon,\phi,\gamma}{\underset{l_1}{\circ}\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\underset{l_2}{\circ}} \quad \Rightarrow \quad \overset{l_1,\alpha,\beta,\gamma}{\underset{l_1}{\circ}\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\underset{l_2}{\circ}}$$

where the node labelled l_1 and l_2 are existing (connected) nodes in an arbitrary graph.

The observations that were noted in relation to the CSM productions also apply here.

5. Modelling the S/R Operations within the NLC-grammars

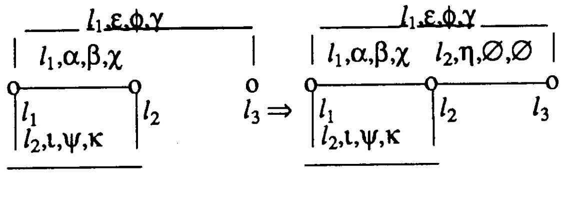
Let H be a graph, with x, y, and z $\in V_H$, $\Psi_H(x) = l_1$, $\Psi_H(y) = l_2$, $\Psi(z) = l_3$.

Before we start to model the S/R operations in the defined grammar, a brief explanation of the edge classification is in order. We said that, given an edge between two nodes labelled l_1 and l_2 with edge context $(l_1, \alpha, \beta, \chi)$, the role of the sets β and χ is to *qualify* certain rights within the set α. We now define these sets to be used as follows: if the *send* right appears within α, the set β contains the set of rights that l_1 may send to l_2. Likewise for χ and the *receive* right: if *receive* $\in \alpha$, χ contains the set of rights that l_1 may receive from l_2. If any other rights appear within α, they have their usual meaning, and need not be qualified.

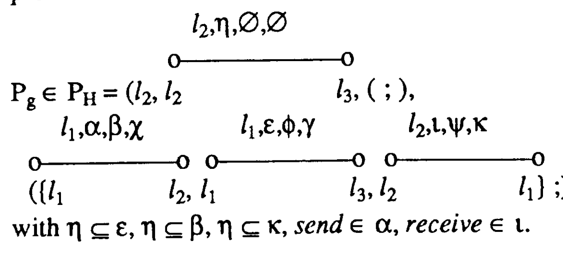
1. Send

Let there be an edge between x and y with edge context $(l_1, l_2, \alpha, \beta, \chi)$ with *send* $\in \alpha$. Let there also be an edge between x and z with edge context $(l_1, l_3, \varepsilon, \phi, \gamma)$ with $\eta \subseteq \varepsilon$ and $\eta \subseteq \beta$. Lastly, let there be another edge between y and x with edge context $(l_2, l_1, \iota, \psi, \kappa)$ with *receive* $\in \iota$, $\eta \subseteq \kappa$.

The send operation establishes a new edge between y and z with edge context $(l_2, l_3, \eta, \varnothing, \varnothing)$. (The reason for the empty sets in this context is that they correspond to qualifiers or rules in the S/R model, and the transportation of these rules is not possible in our formulation of the model). In this production, x sends y the set η of rights, which x has on z, for y to operate on z. The graphical representation of this operation is as follows:



We model this operation by means of the following production:

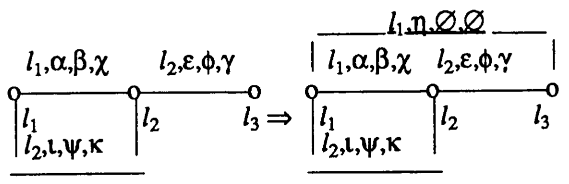


with $\eta \subseteq \varepsilon$, $\eta \subseteq \beta$, $\eta \subseteq \kappa$, *send* $\in \alpha$, *receive* $\in \iota$.

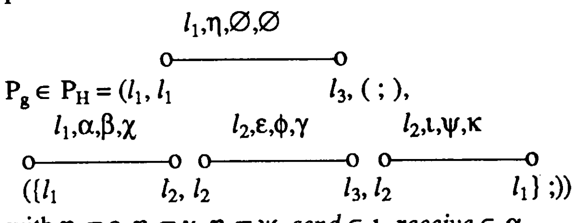
2. Receive

Let there be and edge between x and y with edge context $(l_1, l_2, \alpha, \beta, \chi)$ with that *receive* $\in \alpha$. Let there also be an edge between y and z with edge context $(l_2, l_3, \varepsilon, \phi, \gamma)$, with $\eta \subseteq \varepsilon$, and $\eta \subseteq \chi$. Lastly, let there be another edge between x and y with edge context $(l_2, l_1, \iota, \psi, \kappa)$ with *send* $\in \iota$, $\eta \subseteq \psi$.

The receive operation establishes a new edge between x and z with edge context $(l_1, l_3, \eta, \varnothing, \varnothing)$. (The reason for the empty sets in this context is the same as given above). In this production, x receives from y the set η of rights with which to operate on z. The graphical representation of this operation is as follows:



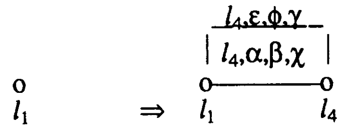
We model this operation by means of the following production:



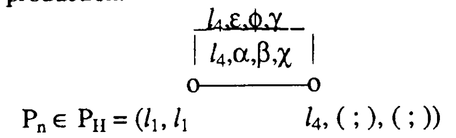
with $\eta \subseteq \varepsilon$, $\eta \subseteq \chi$, $\eta \subseteq \psi$, *send* $\in \iota$, *receive* $\in \alpha$.

3. Create

Let x be a node in a graph H. The Create operation adds a new node n to the graph, with $\Psi_H(n) = l_4$, as well as one edge between x and n with edge context $(l_1, l_4, \alpha, \beta, \chi)$ with $\alpha = R$, $\beta = \chi = R - \{send, receive\}$, and a second edge between n and x with edge context $(l_4, l_1, \varepsilon, \phi, \gamma)$ such that $\varepsilon = \alpha$, $\phi = \beta$, $\gamma = \chi$ (this is done to make full bidirectional transfer of rights possible). This operation is denoted graphically a follows:



This operation is implemented by the following production:



with $l_4 \in \Sigma_2$.

6. Conclusion

There are still some issues to be investigated, and the most important of these would be the development of a grammar suitable for the modelling of protection heuristics, and for the definition of a security model that incorporates the use of expert systems theory.

References

[1] D. Janssens and G. Rozenberg, [1980], On the structure of Node-Label-Controlled graph languages, *Information Sciences*, **20**, 191-216.
[2] R.J. Lipton and L. Snyder, [1977], A linear time algorithm for deciding subject security, *Journal of the Association for Computing Machinery*, **24** (3), 455-464.
[3] N.H. Minsky, [1984], Selective and locally controlled transport of privileges, *ACM Transactions on Programming Languages and Systems*, **6** (4), 573-602.
[4] L. Snyder, [1981], Formal models of capability-based protection systems, *IEEE Transactions on Computers*, **C-30** (3), 172-181.
[5] L.Snyder, [1981], Theft and conspiracy in the Take-Grant model, *Journal of Computer and System Sciences*, **23**, 333-347.
[6] S.H. von Solms, [1984], Node-Label-Controlled graph grammars with context conditions, *International Journal of Computer Mathematics*, **13**.
[7] S.H. von Solms, [1980], Rewriting systems with limited distance Permitting context, *International Journal of Computer Mathematics*, **8** (A), 223-231.
[8] S.H. von Solms, [1982], Rewriting systems with limited distance forbidding context, *International Journal of Computer Mathematics*, **11** (A), 227-239.
[9] D.P. De Villiers, [1986], Logiese Sekuriteits-model gebaseer op NLC-grammatikas, M.Sc. Dissertation, Rand Afrikaans University.
[10] D.P. de Villiers and S.H. von Solms, A logical security model based on NLC-grammars, Submitted.
[11] D.P. de Villiers and S.H. von Solms, Formalizing the Send/Receive security model using NLC-grammars, Submitted.

SOUTH AFRICAN INSTITUTE OF COMPUTER SCIENTISTS

Members in Arrears

Mrs S Berman
Prof C H Bornman
Mr M J Chapman
Mr A K Cooper
Mr W A Cronin
Mr W R Jones
Mr S M Kaplan
Prof P S Kritzinger
Mr D N Lubowitz
Prof K J MacGregor
Mr D J Malan
Mr R J Mann
Mr J C Mamewick
Mr S A Matsoukis
Mr G D Oosthuizen
Mr R P Perold
Mrs A E G Potgieter
Mr C D Reynecke
Mrs C M Richfield
Mr I S Shaw
Mej F Spoelstra
Mr S L Stoch
Mr T Turton
Mr W van Biljon
Prof R van den Heever
Mr W L van Niekerk
Dr D E Wolvaardt
Prof J S Wolvaardt

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Professor J M Bishop
Department of Computer Science
University of the Witwatersrand
Johannesburg
Wits
2050

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review Categories.

Manuscripts may be provided on disc using any Apple Macintosh package or in ASCII format.

For authors wishing to provide camera-ready copy, a page specification is freely available on request from the Editor.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

[1] E. Ashcroft and Z. Manna, [1972], The Translation of 'GOTO' Programs to 'WHILE' programs, *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255.

[2] C. Bohm and G. Jacopini, [1966], Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM,* **9**, 366-371.

[3] S. Ginsburg, [1966], *Mathematical Theory of Context-free Languages,* McGraw Hill, New York.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimise the risk of the author's contribution having to be held over to a later issue.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.