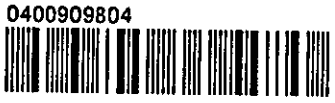


A REFERENCE ARCHITECTURE FOR FLEXIBLY INTEGRATING MACHINE VISION WITHIN MANUFACTURING SYSTEMS

by

John Edwards

A Doctoral Thesis submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy

of Loughborough University of Technology

Department of Manufacturing Engineering

November 1993

Declaration

No part of the work described in this Thesis has been submitted in support of an application for any other degree or qualification of this or any other University or any other institute of learning

Acknowledgements

The author wishes to thank:

Professor R.H. Weston for his help and encouragement.

All Members of the Systems Integration Group at Loughborough for their help and friendship.

Special thanks to Lynn for her patience.

ABSTRACT

A reference architecture provides an overall framework that may embrace models, methodologies and mechanisms which can support the lifecycle of their target domain. The work described in this thesis makes a contribution to establishing such a generally applicable reference architecture for supporting the lifecycle of a new generation of integrated machine vision systems.

Contemporary machine vision systems consist of a complex combination of mechanical engineering, the hardware and software of an electronic processor, plus optical, sensory and lighting components. This thesis is concerned with the structure of the software which characterises the system application.

The machine vision systems which are currently used within manufacturing industry are difficult to integrate within the information systems required within modern manufacturing enterprises. They are inflexible in all but the execution of a range of similar operations, and their design and implementation is often such that they are difficult to update in the face of the required change inherent within modern manufacturing.

The proposed reference architecture provides an overall framework within which a number of supporting models, design methodologies, and implementation mechanisms can combine to provide support for the rapid creation and maintenance of highly structured machine vision applications. These applications comprise modules which can be considered as building blocks of CIM systems. Their integrated interoperation can be enabled by the emerging infrastructural tools which will be required to underpin the next generation of flexibly integrated manufacturing systems.

The work described in this thesis concludes that the issues of machine vision applications and the issues of integration of these applications within manufacturing systems are entirely separate. This separation is reflected in the structure of the thesis. PART B details vision application issues while PART C deals with integration. The criteria for next generation integrated machine vision systems, derived in PART A of the thesis, are extensive. In order to address these criteria and propose a complete architecture, a "thin slice" is taken through the areas of vision application, and integration at the lifecycle stages of design, implementation, runtime and maintenance.

The thesis describes the reference architecture, demonstrates its use though a proof of concept implementation and evaluates the support offered by the architecture for easing the problems of software change.

CONTENTS

Chapter 1

INTRODUCTION

NEXT GENERATION INTEGRATED MACHINE VISION SYSTEMS

1.0 INTRODUCTION

This thesis presents a reference architecture for supporting the design, implementation and maintenance of next generation machine vision systems, where these systems will be flexibly integrated within CIM systems.

The structure of the thesis reflects a fundamental separation between issues pertaining to machine vision applications, and issues pertaining to the integration of these applications within CIM systems. The thesis comprises three principal parts:

- PART A - A DERIVATION OF THE "NEEDS" OF NEXT GENERATION INTEGRATED MACHINE VISION SYSTEMS

  Chapters 2, 3 and 4 present the contemporary position within the disciplines pertinent to the work, namely: manufacturing industry and the role of machine vision; systems integration; and, contemporary practice in machine vision design and implementation.

- PART B - THE DESIGN, IMPLEMENTATION AND MAINTENANCE OF MACHINE VISION APPLICATION OBJECTS

  Chapters 5, 6, 7 and 8 details the techniques proposed for structuring machine vision application software. Their benefit is demonstrated through a proof-of-concept implementation and demonstration of change.

- PART C - THE FLEXIBLE INTEGRATION OF MACHINE VISION APPLICATION OBJECTS WITHIN MANUFACTURING SYSTEMS

  Chapters 9 and 10 detail the techniques proposed for mapping application objects as described in PART B onto an open distributed vision application implemented on an integrating infrastructure. Again, the benefits are demonstrated through a proof-of-concept implementation and demonstration of change.

The final chapter (PART D) draws conclusions from PARTS B and C and proposes future work.

This chapter draws conclusions from the literature presented in PART A and states the "needs" of integrated machine vision. An overall framework which describes the elements required to structure the design and implementation of a new generation of integrated machine vision systems is introduced.

In PARTS B and C, the author identifies and tests the use of models to support design and implementation, methodologies for systematised design, and mechanisms for structured implementation. These models, methods and mechanisms can be used to support the creation and maintenance of next generation integrated vision machines which fulfil the requirements identified within this chapter. These requirements are briefly summarised as follows:

- ease with which structured applications software can be built;
- ease with which structured applications software can be modified;
- ease with which implementation hardware can be changed;
- provision of "designed in" flexibility;
- provision of applications as object modules with defined interfaces and hidden information and functional detail;
- provision of applications that can benefit from available support information;
- provision of applications that generate information from which other manufacturing applications can benefit;

- provision of application-generation tools supported by models and methodologies that enable the requirements above and encourage the generation of structured and flexible applications which support ordered change;
- provision of integration tools supported by models and methodologies to enable application objects to become building blocks of CIM systems, so they can be aggregated within such systems.

It is evident from these requirements that a number of issues need to be studied in order to propose a overall reference architecture for next generation integrated machine vision systems. The work described in this thesis covers machine vision application issues, open systems integration issues, and uses novel, model driven and information generating, implementations in order to define an architecture which supports all the identified requirements. Since the emphasis is on the total system rather than individual components, a "thin slice" is taken through each topic studied.

It is understood that many of the methods underlying the author's work can be found within the design and implementation of contemporary complex software systems. The novel aspect of the author's work lies in the combination of contemporary methods applied to a particular application domain. Within the vision applications work, for example, these methods include object oriented design and implementation, model based design (where the design follows that of the physical structure of the entity likely to require change) and model driven implementation, together with the use of an architectural framework, a vision model and modelling methodology. The derivation of integration methodologies includes the use of a more global architectural framework incorporating the use of an integrating infrastructure, a virtual vision machine and complementary application support element.

2.0 THE REQUIREMENTS FOR INTEGRATED MACHINE VISION

2.1 Introduction

Chapters 2, 3 and 4 of this thesis study the current state of the integration of machine vision systems within manufacturing industry. An important theme throughout these early chapters is

the need for future manufacturing systems, and the individual manufacturing applications that make up those systems, to be able to cope with required change [Weston 90/91/92, Spackman 92, Bishop 89]. The need for this flexibility can be attributed to the following three drivers:

- changes in line with market requirements (market push), meaning that manufacturing industry must remain competitive through the generation of new and updated products [Bishop 89];
- changes made possible through the availability of advanced technology (technology pull), where incorporating new technology within the manufacturing process (and in the manufactured product) can reduce cost and improve the product [Bishop 89];
- flexibility within the production process, e.g. multiple products can be manufactured in batches from a single facility.

The benefits derived from current forms of automation and systems integration within a manufacturing system are identified in chapters 2 and 3. Chapter 3 also identifies contemporary implementation problems within conventional, or "hard" integrated systems which cannot adapt readily to change [Weston 92, Coutts 92]. The benefits derived from integration can enable a company to gain a competitive edge. However, the creation of hard CIM systems can become a barrier to maintaining that competitive advantage. Their resistance to change can prevent the company from employing new technology, and can constrain the design of new products. Hard integrated systems may also lead to inflexible production methods, and systems which cannot change in response to changes in production profiles.

A number of authors have concluded that the ability to embrace change is of paramount importance [Weston 90/91/92, Mertins 92, Spackman 92, Bishop 89]. In the context of applications within information systems, Spackman extends this emphasis on change to illustrate how competitive advantage can be gained by using systems that can cope with change, by rapidly adapting to new market demands [Spackman 92]. The author believes that this potential to profit from change is equally applicable within shop-floor manufacturing systems, particularly if in the future these systems are more like a set of integrated open applications.

Chapter 3 describes the current state of CIM implementation. The existing problems are described together with the proposed solutions which have evolved. The chapter introduces

the notion of "soft" CIM systems, where the elements that make up an integrated system are implemented via an integrating infrastructure providing managed integration services [Weston 92, Coutts 92]. The benefits of this technique include the promotion of standard modular building blocks of integrated manufacturing systems.

Chapter 4 describes the current state of machine vision systems, highlighting the lack of facilities to support their systemised creation and maintenance, relative to the requirements of soft CIM systems. The need for a prescriptive architecture, or framework, for vision machine design and implementation, at least within specific domains, is identified. This architecture could lend structure, and provide guidelines for submodule definition, which could help speed initial system implementation and cope with required change. Chapter 4 introduces the object oriented paradigm and illustrates its potential for the design and implementation of machine vision systems.

The key "need" for integrated machine vision (as identified within PART A of this thesis), is that:

machine vision systems should be designed and built as modular building blocks of integrated manufacturing systems, where their design and runtime architecture should embrace the requirement for ease of change and flexibility

In order to identify the more specific needs of next generation integrated machine vision systems it is important first to identify the nature of the soft integrated manufacturing cell in which such vision processes exist.

2.2 A Soft Integrated Manufacturing Cell.

A schematic representation of a future manufacturing cell is shown in Figure 1, which depicts two distinct views of the cell, a logical application architecture and a physical architecture. The logical architecture structures the functionality of the processes within the cell as a set of open applications running on an integrating infrastructure. These open application objects will pass messages to each other via the integrating infrastructure, thereby making use of the interaction services and information services offered by the infrastructure. The physical architecture will comprise a heterogeneous range of processor-based hardware, the majority of which

are capable of running integration infrastructural software as a layer above the level of their operating system. Chapter 3 identifies the need for handling processing resources that cannot support the integrating infrastructure [Weston 91, Gascoigne 92]. These non-compliant resources must be interfaced with the infrastructure through software principally to provide protocol conversion. This special handling software enables applications running on non-compliant resources to appear as open applications to the rest of the manufacturing applications.

Operator interface facilities for all manufacturing applications within the cell will be through a set of windows on one or more workstations.

FIGURE 1. a conceptual view of a soft integrated manufacturing cell

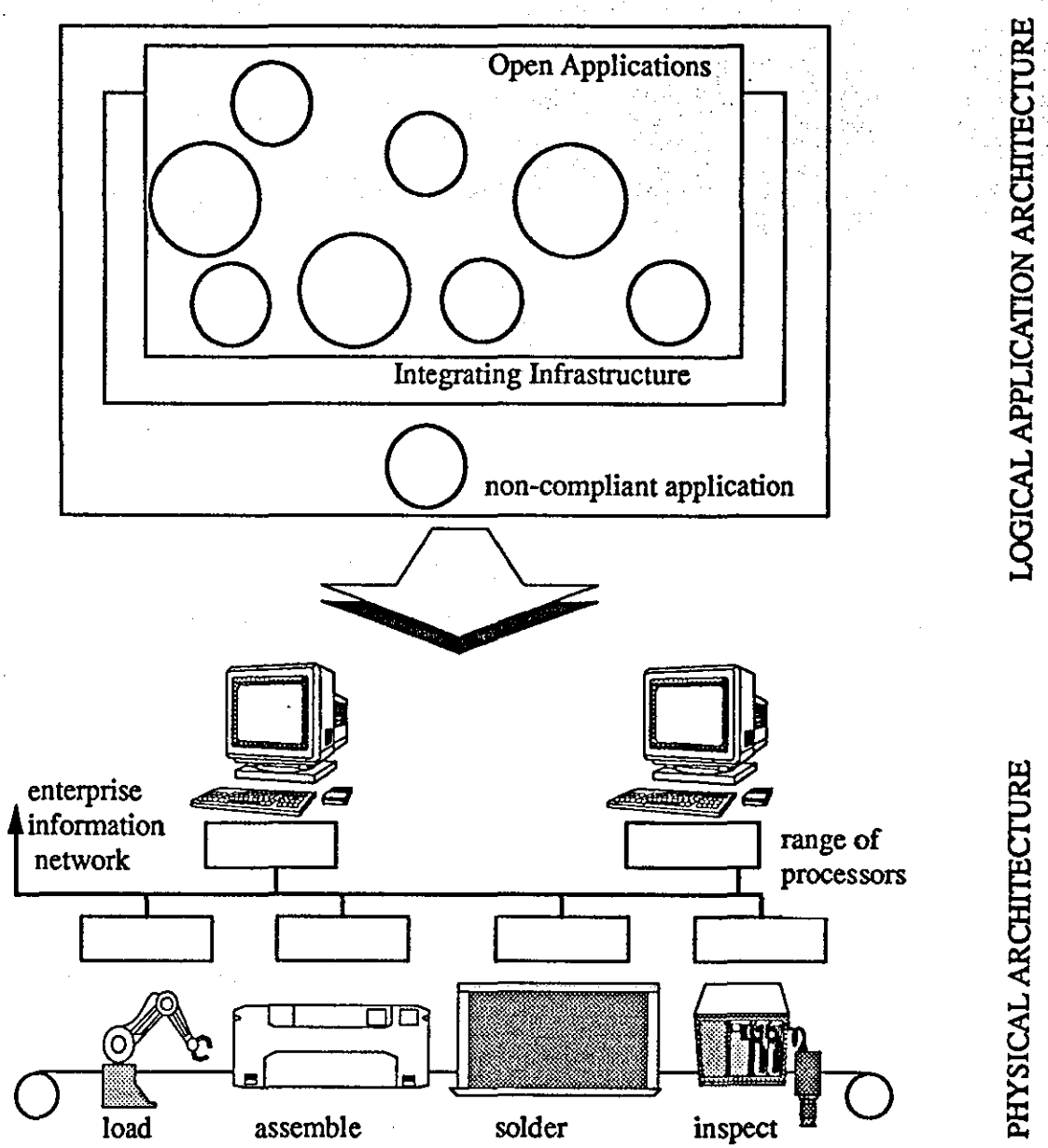
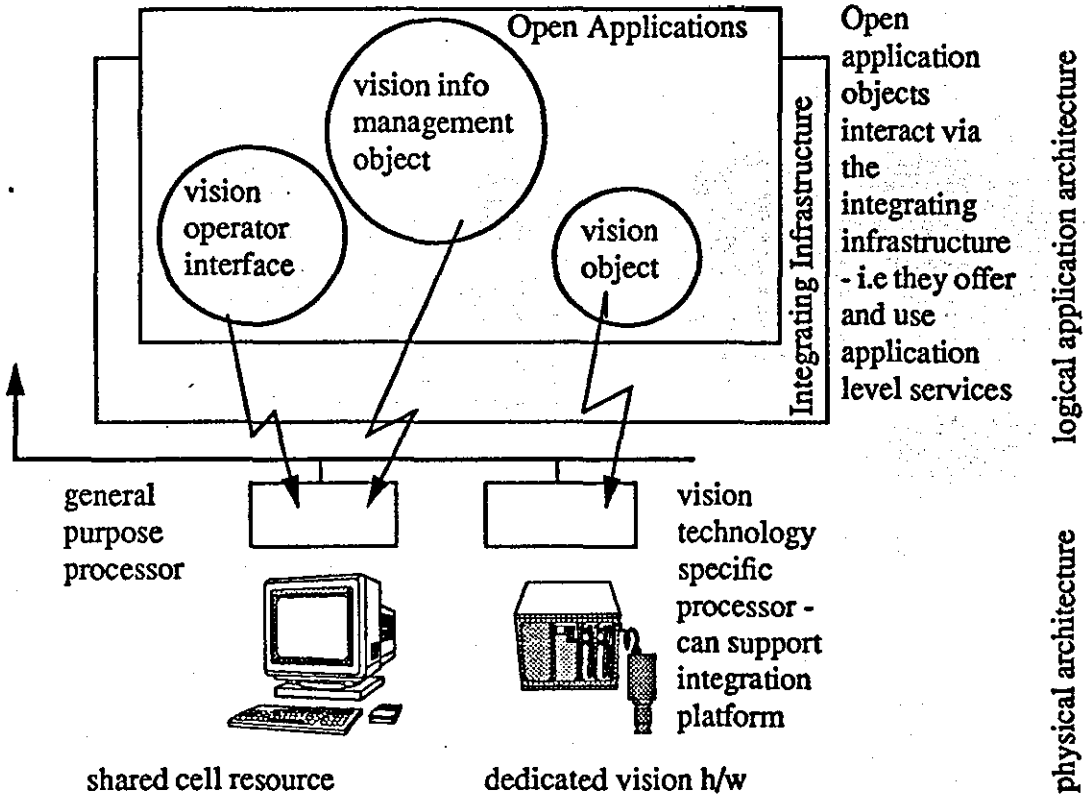


Figure 2 shows more detail of a vision process within a soft integrated manufacturing cell. It introduces the idea of the machine vision application being decomposed into discrete objects which form building blocks for soft CIM systems. This figure shows the open applications being mapped onto various processing components of the physical architecture.

FIGURE 2. A single process within a soft integrated manufacturing cell



2.3 Key Requirements For Next Generation Machine Vision Systems

The following is a set of key requirements identified by the author as necessary for the evolution of integrated machine vision systems. These requirements are derived from a literature survey, experimental work and the discussions with machine vision users and vendors reported in chapters 2, 3 and 4.

- Ease with which structured application software can be built. Chapter 4 concludes the need for application software structured to conform to a reference architecture, where that structure takes the form of rules or fundamental truths relating to a particular technology type or application domain. Chapter 4 also identifies the need for non-prescriptive methodologies based on structured design paradigms.[Thomas 91].

- Ease with which structured applications can be modified. Chapters 2, 3 and 4 highlight the increasing importance of building systems which can support change [Weston 90/91/92, Spackman 92]. Change is an inherent requirement of present and future manufacturing systems and their component elements [Bishop 89]. In this thesis, the term "ease of change" will imply an inherent ability to handle process modification in line with unforeseen requirements (i.e. the required inspection of additional features of a product which were not in the original specification and design of the vision machine inspection system). The implication here is the requirement to handle software modification.
- Provision of flexibility. In this thesis, the term "flexibility" will imply an inherent ability to handle variation within a systems target domain (i.e. shop floor vision machines will be required to handle the changing requirements of a flexible manufacturing line which may be required to cope with multiple options of its manufactured products. In this case the required flexibility can be identified during the design phase of the vision system life cycle.).
- Provision of facilities to enable the removal, replacement, addition, modification or reconfiguration of hardware elements within the system, such that advantage can be taken of evolving technological advance, while application functionality can be retained.
- Provision of object modules which provide specific services so that users of these services do not need to know the implementation details relating to the module elements of the systems to be built. The structure of primary object modules which stand alone within a soft integrated manufacturing system form modular building blocks within that system.
- The ability to make use of available information to support or enhance the functionality of the vision application, typically increasing reliability or flexibility, or even lending a degree of intelligence to an application.
- The ability to contribute information through the provision of open vision services, where this information can be of significant benefit to the applications which make up a manufacturing enterprise [Edwards 93].
- Provision of design and implementation tools to enable the consistent generation of applications in line with the requirements above.

The criteria above have an underlying requirement for the definition of an architecture which potentially provides for rapid implementation of structured systems that supports subsequent software modification. They can be considered in two broad classifications. These will be referred to as platform criteria and application criteria.

- Platform criteria - i.e. those criteria which when satisfied provide a platform for building next-generation vision applications. This class includes:
  - ease with which structured applications can be built;
  - ease with which structured applications can be modified;
  - provision of facilities to support hardware heterogeneity;
  - provision of automated design and implementation tools.
- Application criteria - i.e. those criteria which should be satisfied by vision applications which are built using the tools developed to fulfil the platform criteria. This class includes:
  - provision of flexibility;
  - ability to make use of available information to support or enhance the functionality of the vision application;
  - ability to provide information as an additional resource which could benefit other manufacturing applications within an enterprise;
  - ability to create object modules which provide specific services so that the users of these services do not need to know the implementation details relating to the module.

Some overlap may occur between platform criteria and application criteria i.e. object modules created for a particular application can contribute to platform facilities for rapid build of new applications.

3.0 THE DECOMPOSITION OF VISION APPLICATION ISSUES AND INTEGRATION ISSUES

Figure 3 reflects the fundamentally important decomposition which separates issues associated with the design of application functionality from those issues associated with the implementation of the applications within a CIM system.

FIGURE 3. An application design and implementation environment

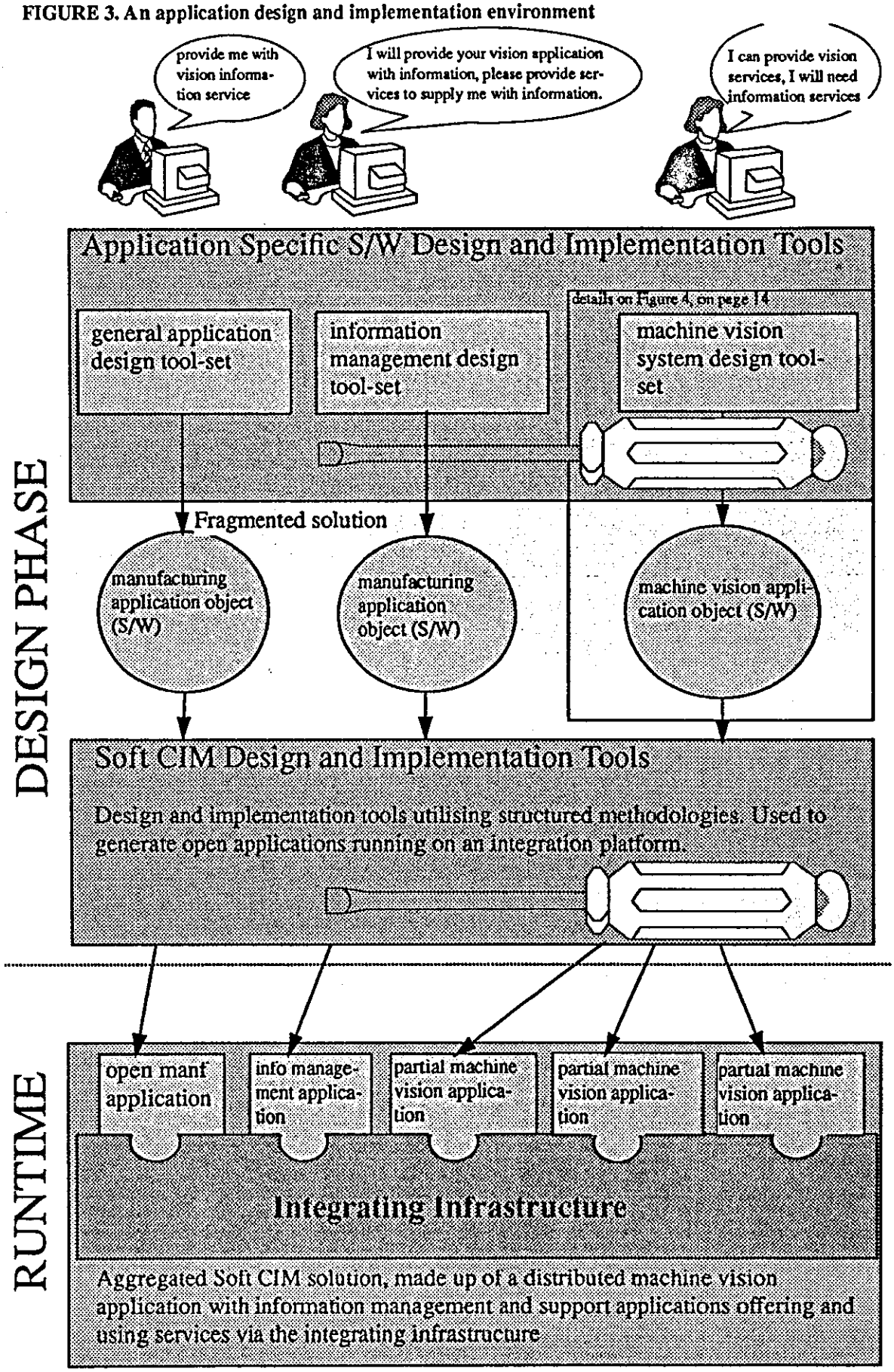
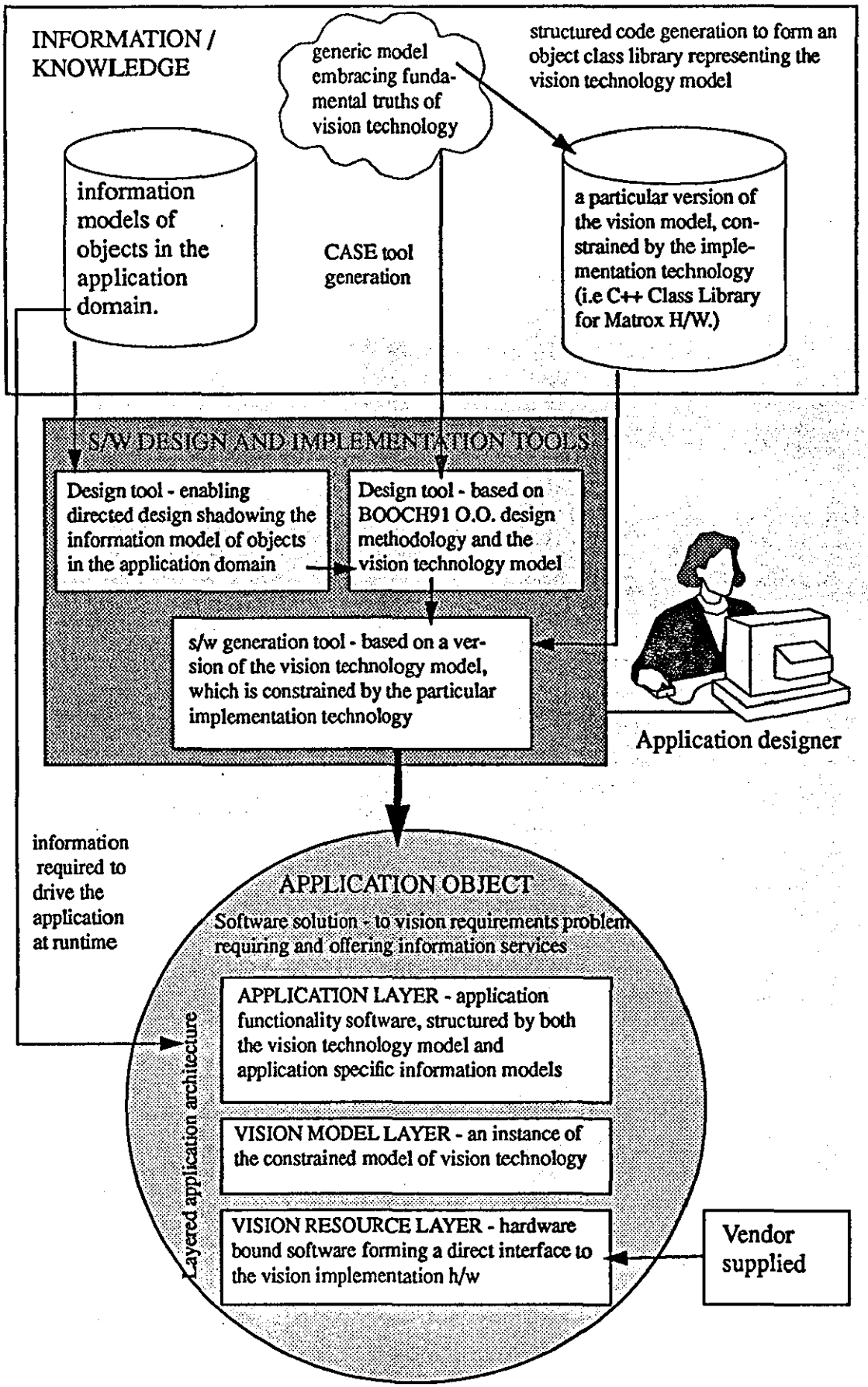


FIGURE 4. The model based machine vision s/w design and generation environment



The tool-set for the design of machine vision systems includes tools to generate a structured, machine vision application. A machine vision application provides vision information services, and can benefit from the support of information contained within the host CIM system. The issues associated with the design and implementation of vision machine applications is considered further in Section 4.0 of this chapter.

Figure 3 also shows a range of fragmented software solutions which could be generated using the application specific design tools. These represent the application functionality (potentially soft CIM system building blocks) required to implement discrete processes, introduced in Figure 1, as application objects. These processes typically could include a vision machine to handle automated inspection, a cell controller to co-ordinate information throughout the production of a printed circuit board, and a maintenance scheduler. At this stage, within the figure, this range of application software solutions are represented as a fragmented set of discrete modules. What is required is that these modules should be implemented so that they co-exist as an aggregated solution offering and using information within a soft integrated manufacturing system. These requirements are considered further in Section 5.0 .

4.0 THE DESIGN, IMPLEMENTATION AND MAINTENANCE OF MACHINE VISION APPLICATION OBJECTS

Figure 4 shows an expanded version of the machine vision design tool-set introduced in Figure 3. It shows a layered architecture to which the discrete machine vision application software solution generated by the design and generation enviroment should conform. A number of important issues are introduced in this figure. These will be discussed briefly at this point to enable on overall understanding of which aspects of the complete system have been investigated by the author and which are reported in this thesis.

The issues addressed within the machine vision application design aspects of this thesis concentrate on the use of contemporary tools to test the notion of defining a prescriptive architecture to provide structure to vision application software and to support change.

The author believes that the functionality associated with digital image processing and feature extraction obeys specific rules and can be modelled as such. This prescriptive vision model can then be used to structure the design of vision application software, particularly the struc-

tured interaction with the software which drives the vision hardware. Figure 4 shows this as a vision model layer within the vision application object.

Structure can be both rigorous and non-rigorous: rigorous structure implies that the software system complies with a particular specification, typically between boundaries within an architecture; non-rigorous structure implies the software system is loosely based on an architecture which lends familiarity to the implementation.

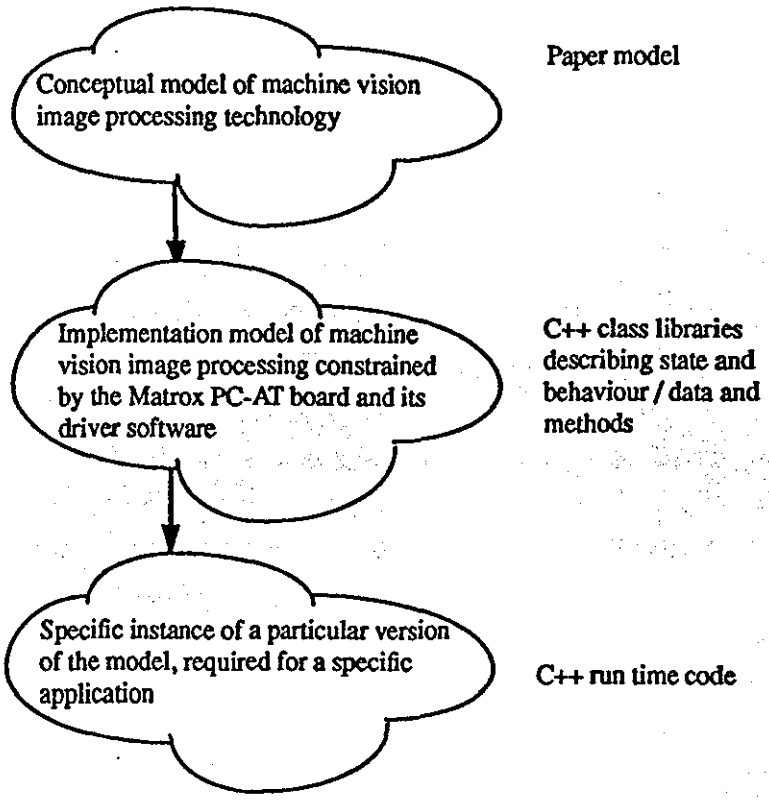
When features have been extracted from an image, the software is entirely application specific and obeys no fundamental or generic rules which can be used to derive a generic reference model to aid rigorous structured design. What is proposed at this level, above feature extraction, is a design methodology and a domain specific framework which can help to provide flexibility and ease of change through non-rigorous structure. This framework would apply in the application layer of the vision application object in Figure 4

Figure 4 identifies four primary elements in the author's proposals for design and implementation of vision application objects:

- "Software design and implementation tools": these tools will be based on, or make reference to, the following models;
- a "vision technology model" embracing both the state and behaviour of vision technology. This model should embrace only the fundamental truths of vision technology that will allow the generation of rules regarding the relationships within the model. Figure 5 shows the stages of the model in isolation, which may help to establish the usefulness of such a model. These stages appear in Figure 4 in the knowledge box at the top of the page and in the vision model layer in the application object;
- "information models" describing a formal representation of the objects which are of interest to a particular machine vision application: the principle of "information driven applications" [Schildt 87], using information to control applications during runtime, is used within this work. This thesis also proposes that the structure of these information models can be used to direct the structured design of applications, using non-rigorous structure to support the creation of modular solution which could enable some of the potential benefits of object-oriented design and implementation to be realised [Cox 87, Booch 91].

- a "layered application architecture". This architecture is used within the structure of the vision application software solution. A simple three layered architecture is proposed, comprising application functionality, vision model and vision hardware resource interface code.

FIGURE 5. the three stages of a model of vision technology I



All the aspects of Figure 4 are covered within PART B of this thesis, though the application designer using CASE tools is replaced by a person aided only by pen and paper with which to implement the proposed structured methodology.

PART B comprises 4 chapters covering the following:

- Chapter 5 - the vision model and modelling methodology;
- Chapter 6 - the use of information models to drive design and runtime systems, and the issues of separation within the three layered architecture;
- Chapter 7 - a proof of concept implementation of a model driven vision application object;
- Chapter 8 - an evaluation of the author's proposals and a demonstration of their support for change.

5.0 THE FLEXIBLE INTEGRATION OF MACHINE VISION APPLICATION OBJECTS WITHIN MANUFACTURING SYSTEMS

The lower sections of Figure 3 show the tool-set which supports the design, implementation and execution of an integrated manufacturing system. This illustrates a scenario in which the software components are open applications running on an integrating infrastructure. This tool-set will have its own support models and methodologies which embrace the life-cycle requirements of a soft integrated manufacturing system. It will include tools to enable the generation of applications as consistent building blocks within a soft CIM system. The scenario depicted within this figure assumes that in the future the open applications software can be mapped onto appropriate processing hardware which is capable of supporting integration infrastructural software.

Figure 6 describes a form of implementation which represents what is currently possible when mapping applications software onto an integrated manufacturing system constrained by the hardware and software used to implement the components of that system. This is in advance of current implementations within industry [Weston 90, Mertins 92, Klittich 90]. The figure describes a scenario where an integrating infrastructure is available and running on a restricted set of hardware and software, typical of the sophisticated systems required for information system networks.

Figure 6 shows a distributed vision inspection application comprising of the following elements:

- the vision processing aspects of the inspection system, implemented as a vision application object providing vision services running on a specialist remote processor;
- A vision client application making use of vision services and providing information to drive the remote vision application object;
- a "specialist application", to be known as a "vision alien device driver", which makes the remote vision application object look like an open application on the integration infrastructure. This module is required to handle all issues of compatibility between the vision application object and the integrating infrastructure. These compatibility issues will always be "one off" problems requiring specialist knowledge as described in the figure.

Each element of the system comprises a layered architecture which reflects the separation of integration issues from application issues. The architecture provides further decomposition to support the separation of application interaction issues from application interoperation issues, where interoperation is supported through techniques based on virtual machines.

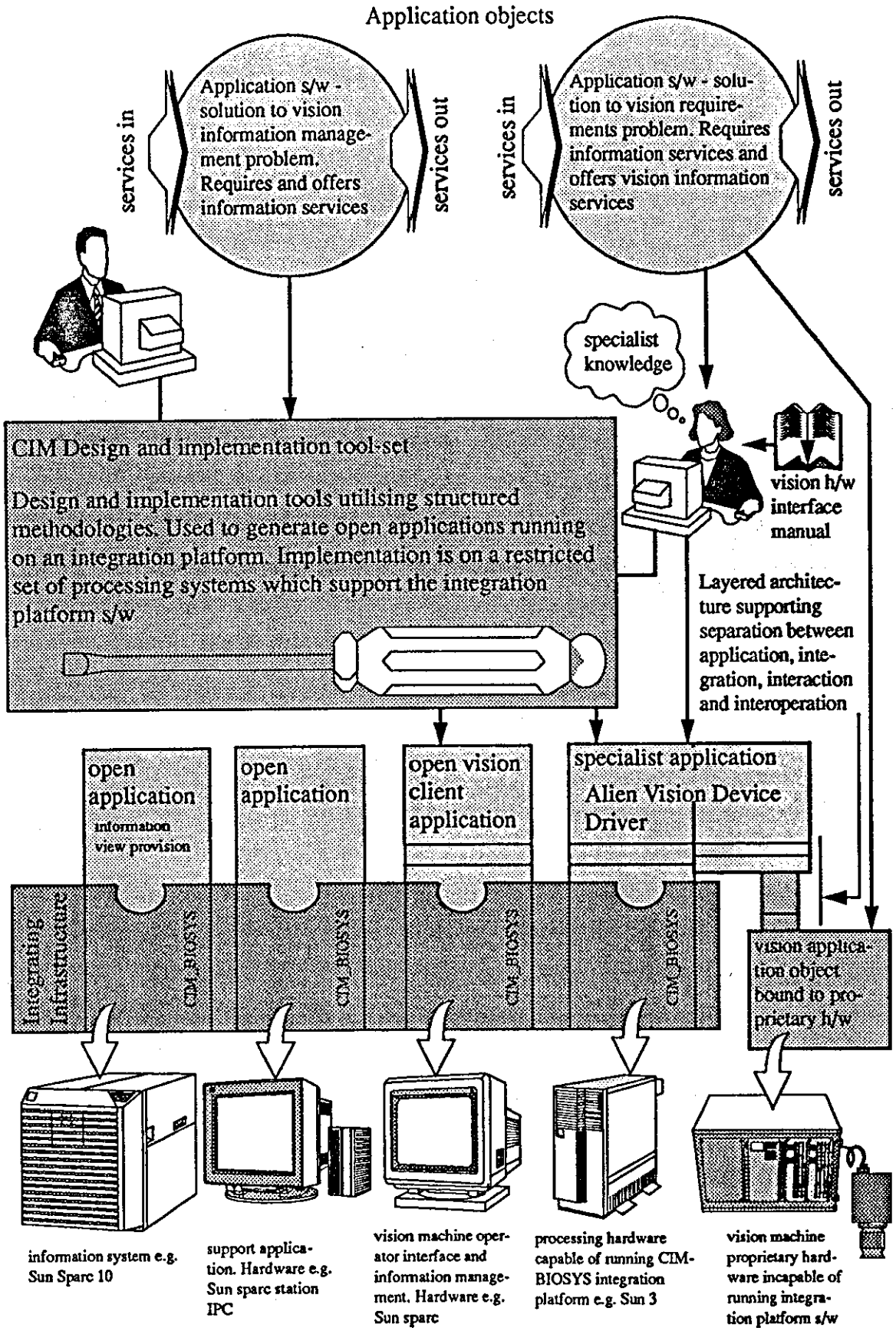
PART C presents details of a comprehensive implementation of a vision machine implemented within a soft integrated manufacturing system enviroment. The system offers a migration path from current hard integrated solutions to the future soft integrated manufacturing systems.

PART C comprises 2 chapters covering the following:

Chapter 9 - a proposed layered architecture required to separate application issues from integration issues, and to further separate interaction issues from interoperability issues;

Chapter 10 - a proof of concept implementation detailing the mechanisms required within the architecture, together with a demonstration of the support for change offered by the author's proposals.

FIGURE 6. The design and implementation of a present soft integrated vision system



PART A

A DERIVATION OF THE "NEEDS" OF NEXT GENERATION INTEGRATED MACHINE VISION SYSTEMS

Chapter 2

TRENDS IN MANUFACTURING AND THE ROLE OF MACHINE VISION

1.0 INTRODUCTION

The first few sections of this chapter concern the global market and its consequences for successful manufacturing. The manufacturing practices which have arisen from the demands of modern economies are identified. The use of information is key. The chapter continues with an outline of the role of machine vision in the electronics manufacturing industry and highlights the need for a new generation of flexible integrated machine vision systems. The potential of machine vision is identified as a provider of valuable and quantifiable process information.

Two underlying requirements which should enable industry to cope with contemporary demands are highlighted, namely:

- infrastructural facilities to enable information access and interaction between manufacturing applications. These facilities would underpin the design and implementation of next generation CIM systems;
- new methodologies for the design and implementation of next generation manufacturing applications, which will use these infrastructural facilities within integrated manufacturing systems.

Fundamental requirements within the specification of this new generation of manufacturing applications are:

- flexibility in terms of short term reconfiguration within a predictable range of variation;
- ease of implementation to enable immediate access to the benefits of automated inspection;
- ease of change to enable the controlled migration from existing technology to embrace future technological innovation;
- ease of integration within CIM systems.

2.0 MANUFACTURING IN THE LATE 20TH CENTURY

In the late 19th century Taylor of the Bethlehem Steel Company in the USA publicised a new philosophy for manufacturing management based on the principles of de-skilling the work-force. This involved setting target times for individual operations, and paying bonuses to those who achieved them [Murray 93, Lipietz 93]. Ford adopted Taylor's principle of timed tasks, and added to it the principle of sychronised flow between operations [Lipietz 93]. The Taylor/ Ford combination produced a philosophy for production and organisation within manufacturing that became generally known as mass production, and it has dominated manufacturing industry for the majority of the 20th century.

Since the end of the second world war, the economies of developed countries have slowly changed. Economies that were demand-led are now dominated by supply [Perrier 92]. From 1945 to 1975 manufacturing industry in those countries responded to economic growth with large-scale production of a single product or a range of similar products, applying the Taylor/ Ford principles of mass production. The late 1960's and early 1970's saw the end of this "boom" period of what had been thought to be unstopable growth [Wee 87]. Since then, the growing sophistication of the global consumer market has meant that manufacturing industry has had to respond to the demand for a diverse and frequently changing range of products [Williamson 92, Hutton 91]. This change has been documented by many observers, and classified by some, as the move from mass production to mass customisation [Davis 91, Hutton 91].

Mass customisation now means small batch sizes and shortened product lifecycles, while intense competition in the global market dictates low unit costs and high quality/reliability. To this end, manufacturing industry has been forced to adopt new methods. Japan has led the world in the practical application of these new methods, and the country has seen a reduction in unit costs as dramatic as that when mass production ousted the individual craftsman [Hutton 91]. For example Horiuchi Kikai Seisakusho, (a Japanese hydraulic cylinder manufacturer) have a product range of 20,000 cylinder types, and offers delivery within four days of receipt of order [Bell 92]. Many new techniques are combined to achieve this level of manufacturing excellence: some of the most widely known and applied are just-in-time (JIT) [Dear 88], Total Quality Management (TQM) [Spenley 91], Design for Manufacture and Test [Katz 91], Statistical Process Control (SPC) [Lochner 90], Advanced Manufacturing Technology (AMT)

[Goetsch 90] and Computer Integrated Manufacturing (CIM) [Ranky 86, Waldner 92]. A practical reference covering most of these contemporary practices is "Ten steps to world class manufacturing" [Steudel 91].

Bishop and Schofield [Bishop 89] identify ten goals for today's successful manufacturer:

- reduced lead time to satisfy customer orders;
- reduced time to get new products to market;
- flexibility to adapt to changes in technology or market;
- increased quality;
- reduced costs;
- increased management control;
- customer satisfaction;
- improved customer service;
- improved use of people;
- meeting safety, environmental and legislative requirements.

They stress that "The only constant is change" in the modern economy, implying that the capability to change must be inherent in the implementation of each listed aim. A modern manufacturing enterprise is characterised by a continuous state of evolution, which embraces new technologies and practices to suit the prevailing economic circumstances. The need to accommodate change is a major consideration when identifying the requirements for the elements which will make up the next generation of integrated manufacturing systems.

Davis and Davidson describe the current economy as the "information economy" in their book "2020 Vision" [Davis 91]. They claim that the new economy has been in existence since the 1950's. However, it was not until the late 1970's and 1980's that there was a general agreement that this fundamentally different economy was in place [Davis 91, Hutton 91, Waldner 92]. Manufacturing industry responded with quality, productivity and customer service initiatives. Mass customisation with rapid response has been an additional requirement in the late 1980's and early 1990's. Davis provides many practical examples of these new manufacturing practices [Davis 91].

Papers describing the application of these new practices such as JIT and TQM conclude the need for a coherent business strategy to identify and position appropriate methods [IBM 87,

Evans 88, Mansel 88]. The need for integration of the business, physically through the use of computers, networks and automation, and logically through the use of information has also been identified [Bishop 89, Weston 88]. It is the combination of these issues that comprises Computer Integrated Manufacturing (CIM) and which is centred on the use of information [Weston 88, Thomson 89].

Thomson and Graefe [Thomson 89] describe CIM as a new paradigm for the "integration of information" which calls for the control of manufacturing through the use of a single master record of all manufacturing information (a theme endorsed by many researchers and commentators in the field [Weston 88, Schofield 88]). They stress that solutions to manufacturing problems no longer spring from increasing the efficiency of each process within the manufacturing cycle, as these no longer represent the significant production cost. For example, at Apple Inc.'s new Macintosh plant direct labour accounts for 1% of production cost, while at IBM it is approximately 4% [Thompson 89]. Manufacturing support now makes the largest contribution to production cost, and it is in this area that integration and information can reduce support costs appreciably.

3.0 THE EFFECTS OF THE NEW PRACTICES WITHIN MANUFACTURING

Figure 1 shows a classical hierarchical structure of a manufacturing enterprise. In the classical structure each department is considered to be a miniature enterprise, and channels of information follow vertical paths between the hierarchical levels. The effect of new manufacturing techniques involves change within all layers of the structure changing the shape of the pyramid, and will involve a reduction in the number of hierarchical levels.

A new type of management structure is emerging to accompany the establishment of integrated information systems within an enterprise. This structure is based on the decentralisation of decision making which breaks down the classical subdivided organisation, shown in Figure 1 [Waldner 92]. In the new structure, communication takes place freely between levels and different specialisms. Vertical lines of communication are not enforced when the progress of the company is involved.

FIGURE 1. The hierarchical structure of a company

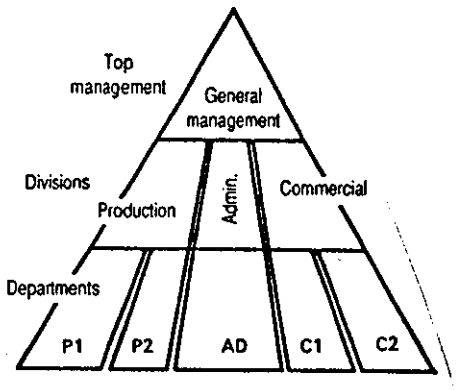


Figure taken from reference [Waldner 92]

This decentralisation requires access to appropriate information to support decision making at every level within the enterprise, and to all entities, be they man or machine, which can benefit from information access.

In order to provide appropriate information throughout an enterprise an infrastructure to support information access is required.

4.0 THE REQUIREMENT FOR AN INFORMATION INFRASTRUCTURE

Davis highlights the importance of providing an information infrastructure, so that business can benefit from the availability of information [Davis 91]. He makes an analogy with other infrastructural systems such as those which exist within the real estate industry where roads, sewage, electricity, gas and telephone need to be established before a community of homes and businesses. Today's data communication networks (which offer an enabling tool which can form part of such an infrastructure) have already had significant impact on the global economy. It follows that in order to implement an integrated manufacturing system based on access to relevant information, infrastructural facilities must first be established.

Within a manufacturing enterprise, another prerequisite for the effective use of information is that entities which can derive benefit from information access are designed to take advantage of the availability of that information. If these entities are human operators or managers, training is a prerequisite. If they are manufacturing applications, a new set of criteria must determine the specification for their design and implementation.

5.0 TWO THEMES WITHIN THE NEXT GENERATION OF MANUFACTURING SYSTEMS

In order for manufacturing applications to make effective use of information a system must provide for the availability of information as and where it is required, and manufacturing applications must be tailored so that they can take advantage of available information.

Weston argues these two principles [Weston 88]:

- "Methods should be studied whereby the activities of currently available heterogeneous computer systems can be integrated to provide improved levels of combined decision making and control functionality. The integration methods would necessarily involve the interconnection of information storage devices (some which would support specific manufacturing applications) and the evolution of integration standards and tools to achieve this interconnection";
- "The need to design a new generation of computer based entities from which manufacturing systems can be built. The new entities would not be designed as stand alone applications but would reflect the need for integration and the enhanced levels of decision making and control functionality which information availability through integration will enable".

The work of the Systems Integration (SI) Group at Loughborough University of Technology (LUT) has focused largely on the work identified in Weston's first point, and this work, along with contemporary world-wide research initiatives, is reported in Chapter 3 of this thesis. His second point was the initial stimulus for the author's work: to investigate how next generation vision machines can be designed and built so that they can be integrated easily into an information infrastructure, and can maximise the advantage of information access. As such, these next generation vision machines will form building blocks within integrated manufacturing systems.

The following section looks at the contemporary role of machine vision within the electronics manufacturing industry, and seeks to identify the future requirements to widen the scope of its effective implementation.

6.0 THE ROLE OF MACHINE VISION

Most manufacturing processes, such as machining and component assembly, can be accomplished using automated machines equipped for high precision tool positioning and proximity sensing. However, inspection operations within the manufacturing industry normally depend on vision [Moir 89]. Moir claims the following benefits from automating visual inspection:

- objectivity;
- 100% inspection;
- quantifiable feedback data on production efficiency;
- elimination of human error and fatigue;
- continuous operation.

The successful application of automated visual inspection within the electronics manufacturing industry has demonstrated many of these benefits [Stroebel 89, Black 88, Oughton 88, Voss 89, Bracker 89, Powell 89].

During recent years the electronics manufacturing industry has had to cope with rapid developments in semiconductor technology, while at the same time having to adapt to changes in manufacturing philosophy required within the new global economy. During this period, machine vision has become an integral part of the automation used to increase the efficiency or quality of the printed circuit board (PCB) fabrication and assembly processes which are required for the new miniature technology.

In many cases vision has been the only effective solution. The PCB fabrication process is by nature inexact, being very sensitive to changes in environmental conditions [Wright 90]. As PCB features have decreased in size the inexactness of the process has become more significant, and it can only be overcome by the use of flexible process machinery which can compensate for every PCB being unique. Machine vision which inspects and feeds back visual information, provides this flexibility. This is not an attempt to "inspect out" avoidable process problems, it is the use of vision to compensate for a process that is inherently inexact.

7.0 CURRENT APPLICATION AREAS FOR MACHINE VISION IN PCB FABRICATION AND ASSEMBLY

7.1 Successful Applications

There are many uses of machine vision in PCB fabrication and assembly [Edwards 90]. During the processes involved in PCB fabrication and assembly, position feedback is used for the realignment of PCBs and their phototools. Realignment follows visual inspection of fiducial marks to provide information about their exact position. Manufacturers who produce a range of contemporary SMT assembly machines incorporating these features include both Fuji Machine Mfg. Co. [Fuji 89] and Panasonic Factory Automation Co. [Panasonic 92].

The fabrication of mulilayer fine line technology PCBs requires that individual layers are inspected prior to their incorporation within a PCB. Because of the miniature scale of the PCB features, testing for the continuity of each track is no longer sufficient. Quantitative measurement of features, such as track width, track-to-track space, track-to-pad space with additional inspection for faults such as pin holes and spurious metal [Bracker 89], is now required to ensure high quality and reliability of finished boards. Automated optical inspection (AOI) together with operator verification has been successfully employed in this task. Typical equipment manufacturers include Optotech, and Lloyd Doyle [Optotrec 89, Lloyd 89]. The inspection of PCB phototools is also carried out using similar AOI machines.

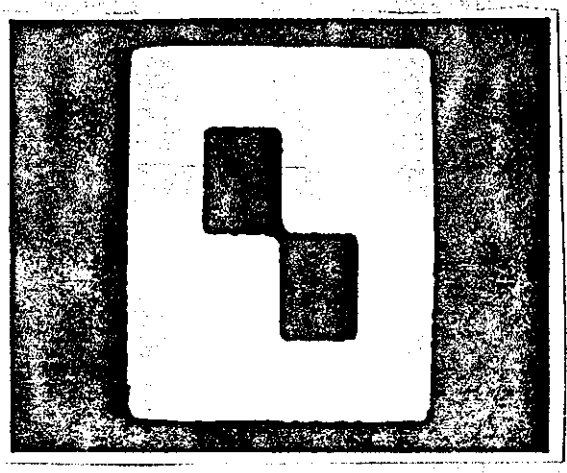
7.2 The Basis for Successful Implementation of Machine Vision

In his paper "Machine vision and its integration with CIM systems in the electronics manufacturing industry" the author has identified some of the reasons for the success of machine vision in PCB fabrication and assembly [Edwards 90]. These include the following:

- when generating offset information for a placement head, component position detection can be achieved by processing the binary image of the device outline. This is commonly achieved by back-lighting the device to produce a black object on a white background. This means process variations such as device colour, surface finish or position and content of lettering on the device can be ignored, as can the problems of extracting the relevant object from a confusing background. This enables the vision machine to have a consistent and fairly simple image to process;

- fiducial marks on the PCB are specifically for the use of the vision machine and are therefore designed for ease of image capture and analysis. Figure 2 shows an example shape that has been used as a fiducial mark, and has proved to produce consistent position information through automated image capture and processing;
- in AOI systems success can be attributed to the good definition of object of interest against background i.e. the tracks and pads against the PCB laminate or artwork film. Thus the subject lends itself to consistent image capture. Figure 3a shows a photograph of a PCB pad and track on laminate, while Figure 3b shows the AOI captured image.

FIGURE 2. Binary image of a fiducial mark captured using front illumination



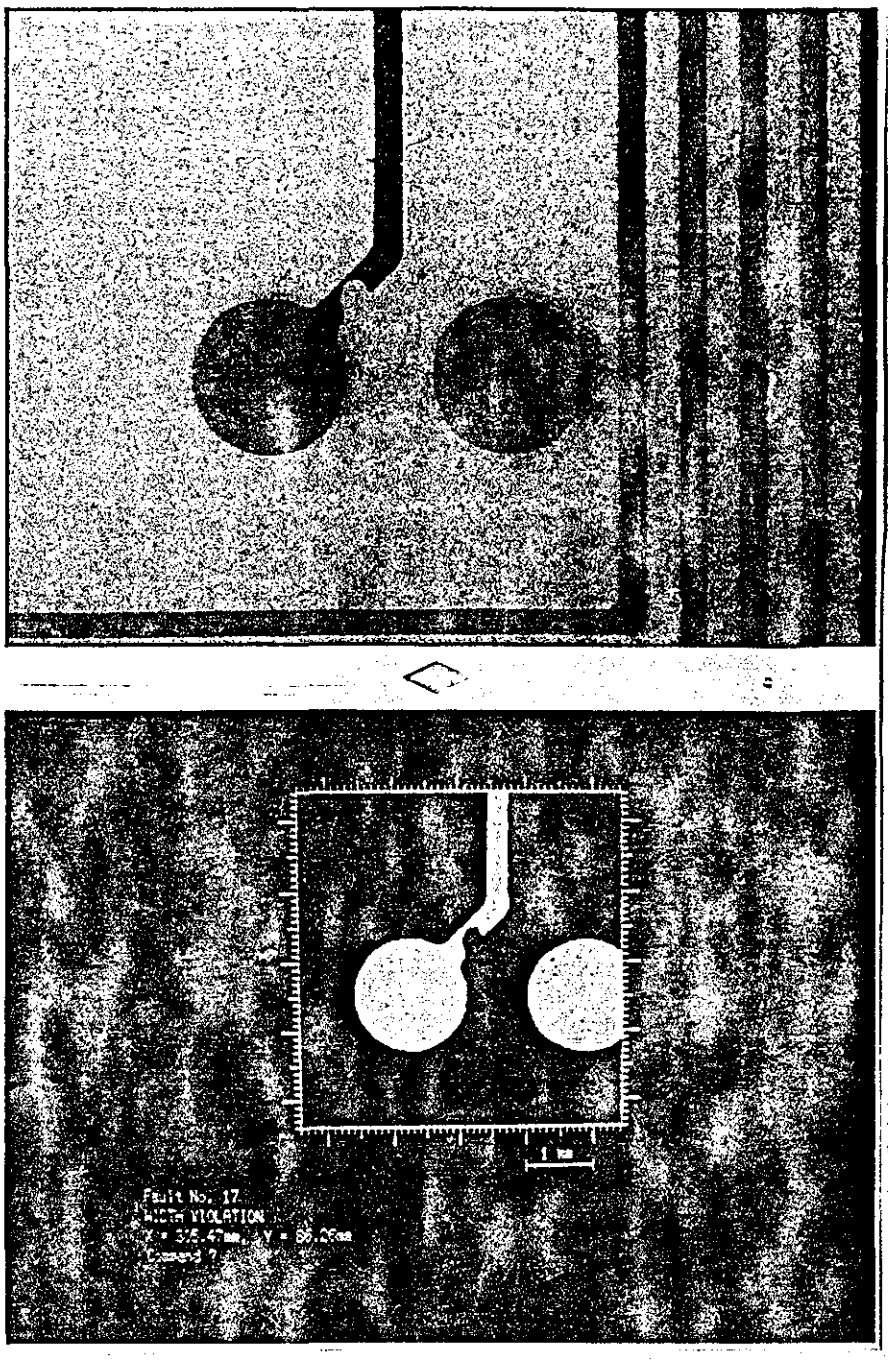
The success of these applications can then be attributed to:

- the design of the product such that the effect of process variables is minimised;
- the ability to simplify the vision task, leading to increased operating speeds and reliable operation.

8.0 PROBLEMS WITH APPLYING MACHINE VISION DURING PCB ASSEMBLY.

The previous section has specified particular niche areas within electronic manufacturing where machine vision can be justified, or indeed, is essential. This section relates some experiences of industrial users of machine vision applied to the more demanding area of inspection during PCB assembly where success has been more difficult to achieve.

FIGURE 3. PCB inspection using Automated Optical Inspection AOI



a. A Photograph of a PCB pad and track

b. An image of the pad and track above captured using an AOI system

8.1 An Experience in the UK

During the late 1980's, ICL (UK) Ltd., which includes a large PCB fabrication and assembly operation at Kidsgrove, were looking to machine vision to provide automated inspection in a number of areas [ICL 89, Edwards 90]. These included final inspection of SMT PCBs prior to electrical test. They feared similar problems to those they had experienced on their through-hole technology automated inspection system. From their experience of running the through-

hole system for several years, ICL estimate a thousand lead PCB will take at least two days of hard laborious work to set up.

Having set up the system, the consistency of the pass/fail classification is often poor. Slight changes in production profiles, such as lead clinch angle, lead crop height or height of the flow solder wave, can alter the appearance of the expected image, and the simple vision system cannot adapt. The system can be made to work, but it requires the regular attention of a technician to adjust it, providing feedback between process variation and inspection criteria. In effect, the technician transfers information between programmable machines.

8.2 An Experience in the USA

The experiences of a manufacturer in the USA serve to demonstrate what can be achieved using machine vision for inspection during PCB assembly.

NCR, a large computer manufacturer, employed eight inspectors per shift for pre-solder inspection of SMT placement [NCR 89, Edwards 90]. Problems included speed, operator fatigue, and inconsistency. In 1987 machine vision inspection equipment was incorporated into the SMT assembly process for pre-solder alignment inspection. Then followed a two year period of development including modification to the inspection equipment, and extensive trials and tuning in order to generate a set of inspection classification algorithms to enable reliable inspection. The completed system included a suite of auto program generation software to enable CAD data to be used to generate vision machine inspection data for new or modified PCBs. The system is also linked to two rework stations where fault data is down loaded and faults rectified by the two inspectors who now man each shift.

The success of the system can be attributed both to its ability to achieve high classification reliability and to the degree of flexibility, in terms of program generation and reduced set-up times. This flexibility is based on the integration of the system within the software tools and programmable machines which make up the design and manufacturing process.

This application also demonstrates the speed at which technological change takes place. Contemporary practice since 1990 is to use vision-assisted placement of SMT components, as described earlier. Although a reliable and flexible system was generated at NCR over the two

year period, by the time it was effective the assembly technology they used was effectively out of date. Contemporary assembly machines would have reduced the need for 100% post solder inspection.

Thomas [Thomas 91] stresses the same general point, that "short product life-cycles typical of modern manufacturing industry presents a very great problem to any manufacturer who needs to exploit automation in general, or computer vision in particular, in order to achieve desired production efficiency". This implies the need for:

- structured methodologies for the rapid implementation of vision machines, and;
- an architecture for vision machines which could enable structured design, implementation and modification in-line with required change.

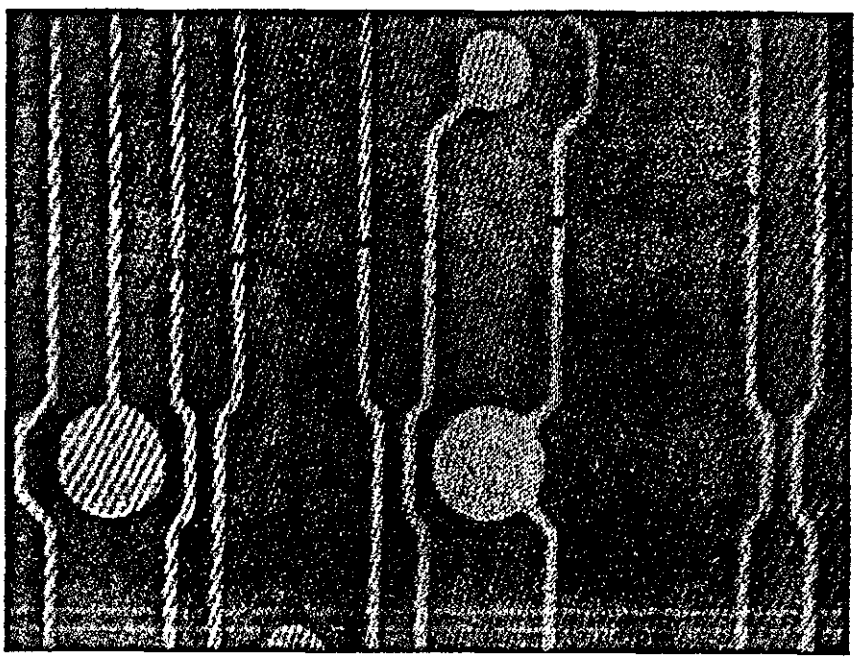
9.0 A VIEW OF THE FUTURE DIRECTION OF AUTOMATED VISUAL INSPECTION WITHIN PCB FABRICATION AND ASSEMBLY

Robert Wright of the Gerber Scientific instrument Company identifies three problem areas in integrating AOI systems so that they can be supported by information generated during the PCB design process [Wright 90]. The first two are the commonly recognised issues of storage of large volumes of data and its compression and decompression. The third issue supports one of the principals presented in this thesis: that integrated vision machines can generate information which could be of benefit to other manufacturing processes, and that this information should contribute to a dynamic information store describing the PCB product (this notion is detailed in chapter 6). Wright states that "the third obstacle to the implementation of a useful CAM reference AOI system is the ability of the system to cope with minor shifts in data caused by process variables". The PCB manufacturing process generates distortion, through innerlayer shrinkage, movement and shift, and through film movement caused by minimal changes in temperature and humidity. These shifts mean each PCB is different from the exact design datum information but do not preclude the correct functioning of the PCB. Wright suggests "this movement or change in the panel data must be observed and quantified, and then a correction made to the database".

Douglas Blakeslee, the Vice President of Applied Vision Systems, also stresses the need for integration of vision processes within information systems on the shop floor [Blakeslee 89]. He identifies the need to store information generated by AOI systems so that it can be made available to the verification process where human operators can verify faults identified by the vision machine. They can also classify the cause of these faults and so feed back information to control the manufacturing process. Figure 4 shows a photograph of the way AOI can relate faults and identify process errors [Cohen 91].

The notion of using information generated via automated optical inspection is supported by [Zwern 90]. Zwern points out that AOI measures two of the basic PCB process variables: how far track widths are from the centres of their tolerance range, and how far pads are from their design positions. In order to make effective use of this information, Zwern advocates increased integration of AOI information with other process data and states "the AOI industry must let go of its outdated closed architecture mentality".

FIGURE 4. Result of AOI on a fine line PCB



An AOI system can classify this defect in two ways: as multiple opens when analysed for quality control or as a fibre introduced at the exposure or resist stage when examined for process control.

Figure taken from reference [Cohen 91]

Many further examples of the PCB manufacturing industry's desire for the integration of vision inspection machines within information systems for both shop floor support and management information can be found in the journals that support the PCB fabrication and assembly industry [Doyle 90, Doyle 91, Chen 90, Danon 91, Gilutz 91].

In their vision of the PCB manufacturing shop in the year 2000, Flat and Holden predict "Visual inspection and verification as it is today will be virtually eliminated. Automated Optical Inspection will be used extensively, not so much for detecting defects as for quantifying parameters. Instead of being clustered at final inspection they will be placed throughout the shop and used to feed back quantitative data" [Flat 91].

10.0 NEXT GENERATION VISION MACHINES

Although automated visual inspection has been implemented successfully in the electronics manufacturing industry, its principal application has been to tasks which it is capable of carrying out more accurately, reliably, and efficiently than a employee, or where the task simply cannot be done by a human operator [Thomas 91]. However, today's climate of short product life-cycles, short production runs, fast times to market, multiple and custom products produced in a fast and flexible manner, together with rapidly changing technology, implies a new set of criteria for effective automation. Considerations such as configurability, flexibility, speed of implementation, ease of update and change of manufacturing process machines are necessary to the successful implementation of the next generation of vision machines, as is ease of integration of these machines within manufacturing information systems [Zwern 90, Flat 91, Doyle 91, Thomas 91].

Contemporary automated visual inspection machines are primarily complex pieces of mechanical engineering with associated electronic programmable controllers. They are designed for a specific purpose as stand-alone processes capable of being bought "off the shelf". They can be installed and put into operation without necessarily being linked to the complex communications and information systems commonly used in their host environments. These vision machines are built for a specific purpose or class of applications with configuration facilities to enable multiple products of the same type to be processed. As such, the main design priorities are software execution speed, reliable vision processing and mechanical

accuracy. Unfortunately, when viewed as a building block of a manufacturing system, these vision machines are generally not flexible and do not offer a migration path which supports modification or change, in line with changes in enabling technology or changing product requirements. [Azar 88, Zhang 92, Edwards 93]

This thesis proposes that the next generation of vision machines can benefit from information support and can provide process information as a contribution to the dynamic information pool which will support the manufacturing cycle. This requirement for information access implies a need for easily integrating vision machines into the information infrastructure of an integrated manufacturing system. Next generation vision machines need to be designed and built using a modular systems approach so that they can combine speed of design and implementation with quality and reliability. They must also provide the flexibility required to support technological change. This implies both a top down approach through systems analysis, design and software implementation, and a bottom up approach through knowledge gained from practical machine vision implementation. This knowledge could be accessed through a model which describes vision technology, and through a library of reusable software code elements based on this vision model.

Chapter 3

MANUFACTURING SYSTEMS INTEGRATION

(and the need for an integrating infrastructure)

1.0 INTRODUCTION

This chapter presents two perspectives of Manufacturing Systems Integration: the conventional view embodying bottom up implementation by linking "islands of automation" such as CAD, CAM and MRP II, and; a contemporary, and as yet conceptual, view involving the top down analysis and design of integrated manufacturing systems from an enterprise-wide perspective.

The chapter identifies the problems of current approaches to CIM. Potential solutions to these problems are identified as:

- the top-down modelling of an enterprise from its business processes to the functional elements which fulfil the requirements of the business, and;
- the bottom-up solution to the implementation problems of integrating and executing the manufacturing applications which make up the functional elements of an enterprise.

A review of the work within the S.I. Group at LUT and of the CIM-OSA project within ESPRIT details the need for an infrastructure to underpin the integration and execution of next generation manufacturing systems. Examples of parallel work within the IT community describe the proposed provision of enabling mechanisms for open distributed processing. This next generation of integrated manufacturing systems is defined as "soft integrated manufacturing" and its benefits are identified. These benefits include the promotion of standard modular building blocks of soft integrated manufacturing systems.

The new generation of "soft integrated machine vision systems" proposed in this thesis make use of the integration tools detailed in this chapter. These new systems can form a modular

building block of a soft CIM System, deriving benefit from, and providing services to the integrated system.

2.0 THE CHANGING SCOPE OF CIM

The term CIM is generally attributed to [Harrington 79]. He describes CIM as the integration of CAD, CAM and MRP II, a description which was supported a decade later [Schnur 87, Cheng 90]. In the early 1980's, the term CIM had been corrupted to mean many things to many people. In particular it was interpreted as the logical linking of CAD and CAM [Allderdice 85, Bunce 88], a link between two of the "islands of automation" within an enterprise. This theme of linking islands of automation was further corrupted as the development of mechanisms to provide physical communication links and logical data interaction between groups of processor based functionality took place without the planning required to implement shared information. This concentration on the communications implementation aspect of CIM can be traced to the limitation of a novel concept, with no relevant standards and very little specific enabling technology.

Standardisation initiatives for communications (typically MAP/TOP [MAP 88, Hollingum 86]) and information interchange (typically IGES [Palfremen 90], PDES [PDES 87], EDIF [EDIF 90]) have made a significant contribution to enabling CIM solutions while further work on such standards will continue to influence progress. However, during the Mid 1980's these standards initiatives were often perceived as the total CIM solution: "manufacturing industries have fallen on the MAP bandwagon as if solving the communications problems would suffice to implement CIM" [Morris 87].

During the late 1980's the definition of CIM was developed to encompass the integration of all elements of a manufacturing enterprise [Bunce 88]. Particular emphasis was placed on planning integration from a business perspective [Bunce 88, Mansel 88, Ralston 87] and the importance of information as an enterprise resource [Eustace 87, Weston 88]. A re-definition used within the Digital Equipment Corporation (DEC) [Wellhoener 88] is typical of such late 1980's CIM theories: "CIM is the process of integrating information, automation and organisation, uniting the entire manufacturing enterprise".

This wider view of the scope of CIM [Bunce 88, Mansel 88] has recently developed further with the release of a reference open system architecture for the building of a computer supported and integrated manufacturing enterprise, namely CIM-OSA (CIM-Open System Architecture [CIM-OSA, Kosanke 91, Wiendahl 92]. CIM-OSA is the product of ESPRIT Project 688, the European Computer Integrated Manufacturing Architecture (AMICE) which was launched in 1986 and is considered further later in this chapter.

3.0 CONVENTIONAL CIM IMPLEMENTATIONS

3.1 Introduction

The need for integrated manufacturing solutions, and their potential benefit to industry, is now accepted. The conceptual scope of CIM systems has developed and will continue to grow, typically to provide increased support for human integration [Yim 92, NATO 92]. However, usable implementations of integration technology have always lagged behind the concepts, and despite high levels of investment, there are few successful wide-scale integrated systems within manufacturing enterprises [BICS 87, Eustace 87, Weston 92]. Reasons for this can be identified by considering the inherent problems and constraints associated with conventional system integration techniques. These create systems which fall into the following two categories [Weston 92, Coutts 92]:

- Off-the-shelf or "Turnkey" systems.

  These systems are particular solutions to well defined situations, and very often support the view that CIM comprises a CAD to CAM link. These systems are highly targeted, of narrow scope and limited flexibility. Integration problems occur when the user wants to incorporate manufacturing applications from a range of vendors whose products are not compatible with the users existing system;

- Bespoke Integrated Manufacturing Systems.

  These systems are tailored to specific requirements. They encompass both the limited scope systems typical of those implemented in the early and mid 1980's built around multi-vendor CAD to CAM links, interfacing "islands of automation", and more recent implementations based on information integration through common database facilities [Ralston 87]. These systems can offer greater benefit than turnkey systems since they incorporate multi-vendor

sub-systems, but they have considerable drawbacks. They are expensive, due to the high-level of implementation-specific systems engineering and they have long implementation lead-times. The bespoke nature of the software within the implementations can also prohibit full achievement of the original required functionality and inhibit required modification and evolution of the system.[Bessant 88, Zhang 92, Coutts 92].

3.2 Key Problems Within CIM

Conventional CIM implementation technology, used in building turnkey and bespoke systems, has offered little but general purpose software tools (e.g. programming languages, operating systems, analysis and design methodologies) to combat the problems of CIM. Key problems within the design and implementation of CIM systems can be classified as follows.

- Complexity: as the scope of CIM systems has increased, the complexity of the integration problem has grown dramatically [Waldner 92]. System components from a spectrum of types and source of manufacture require a variety of styles of interaction and information exchange. An approximate square law increase in complexity can be expected with the increase in number of subsystems to be integrated [Weston 91].
- Separation of integration issues: Integration issues are seldom recognised as distinct from their associated industrial process issues. CIM Solutions which do not incorporate a clear separation between integration and particular process problems are very difficult to modify in line with inevitable required change [Weston 91].
- Usable tools for top down CIM analysis: Research and experience to date has indicated that top-down analysis and design driven by the business requirements of an enterprise is required to provide a structured framework for CIM implementation [Kosanke 91, Young 91, Wiendahl 92]. However the methodologies required are immature, particularly in relation to dealing with the practical constraints of CIM system implementation [Weston 91].
- Standards: Standards to support integration implementation are the subject of initiatives throughout the world (IGES [Palfremen 90], PDES [PDES 87], STEP [STEP 89]), MAP/TOP [MAP 88], EDIF [EDIF 90]). However these standards are still emerging, and are insufficiently complementary, detailed or stable to encourage their widespread use [Weston 91].

3.3 Hard Integration

Turnkey and Bespoke systems do not *overcome* these problems, they exist *despite* these problems. They cannot combat complexity, but use one-to-one interfacing for subsystems communication [Munton 91] as a one-step solution to a specific problem, and as such will always be of limited scope. All knowledge of integration issues is hard-coded within these systems, and can be described as "hard integrated" [Weston 92, Coutts 92].

The primary limitation of hard integrated systems is their inability to adapt to required change. This requirement to adapt to changing demands is essential if an enterprise is not only to gain benefit from a CIM implementation but also maintain this benefit [Spackman 92].

4.0 A CONTEMPORARY VIEW OF CIM

The top down approach to CIM, embodying system analysis and design from a business perspective, is recognised as the only sensible way to decompose and understand the complexities of a manufacturing enterprise [Evans 88, Ralston 87]. The support required for this analysis and design phase is the focus of current work throughout the world [CIM-OSA, MDC 91]. However, the requirements for creating, executing, managing and maintaining CIM systems, are usually approached bottom up [Weston 92, Mertins 92].

Whether senior management have instigated top down CIM analysis and design to create an integrated manufacturing enterprise, or whether a manufacturing department are trying to implement an integrated manufacturing cell, contemporary research has identified the need for infrastructural facilities to support integrated manufacturing. Sections 5.0, 6.0 and 7.0 review elements of this research and conclude the need for an integrating infrastructure to underpin the configuration, execution and maintenance of integrated manufacturing systems.

5.0 INTEGRATION TOOLS AT LUT

5.1 Introduction

The SI Group has produced software tools which provide mechanisms for supporting integration solutions. These tools are in daily use both in research institutes around the world [Gilders 92] and on the shopfloor at ICL Kidsgrove in Staffordshire [Zhang 92].

The culmination of the SI Group's work to date, is the CIM-BIOSYS integrating infrastructure, and support tools, which are used within this work. Figure 1 shows the evolution of CIM-BIOSYS. The research methodology has been centred on a bottom up approach, where a variety of integrated manufacturing systems have been built, and their characteristics evaluated.

5.2 An Integrating Infrastructure To Support 3 Classes Of Integration Service

By the late 1980's the SI group had evolved the notion that manufacturing integration on an enterprise wide scale required support for three classes of problem: data communications; application interaction; and, distributed information management. Not only was it necessary to provide mechanisms to support the resolution of these problems, the support mechanisms themselves required an infrastructure such that they could be structured and managed during the life cycle of an integrated system.

This led to the evolution of the CIM-BIOSYS software suite [Weston 92, Gascoigne 92, Zhang 92]. This software provides integration services to support the three fundamental integration requirements, namely:

- services to enable functional interaction between manufacturing applications, based on OSI layer 7 application association based interaction [Pimentel 90];
- services to enable information access for manufacturing applications, based on both a simple flat file management system and a comprehensive distributed information management system [Clements 92]. The distributed system treats information as objects providing access via application view provision facilities based on the 3 schema approach [Clements 91];
- Services to enable data communication, based on standard communication protocols and computer networks.

The services and their configuration and management facilities form an integrating infrastructure which supports the creation, management and maintenance of an integrated system. Section 8.0 of this chapter gives a more detailed treatment of the CIM-BIOSYS integrating infrastructure.

FIGURE 1. The evolution of Integration Tools within the SI Group AT LUT

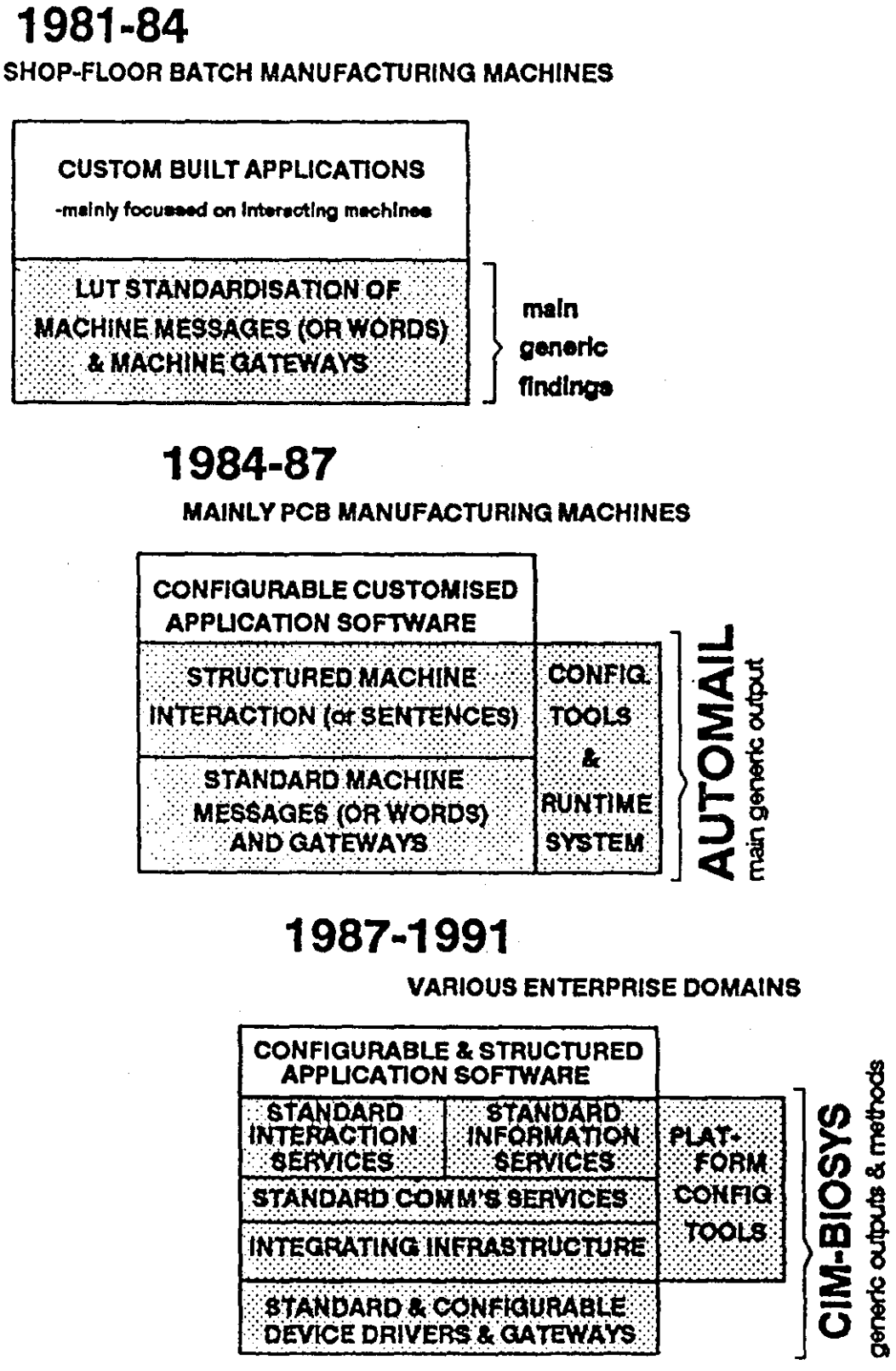


Figure taken from reference: [Weston 92]

6.0 THE CIM-OSA REFERENCE ARCHITECTURE

6.1 Introduction

The CIM-OSA research program has run in parallel with the work at LUT, taking a top down perspective. The CIM-OSA work concludes a similar need for an integrating infrastructure to execute integrated manufacturing applications.

Kosanke [Kosanke91], describes the goal of CIM-OSA as "The provision of an open reference architecture which supports the definition, development and continuous maintenance of a consistent architecture and its related operational system for a particular enterprise". The architecture is intended to allow the modelling, simulation and real time control of all internal and external information needs of a total enterprise. To fulfil these goals CIM-OSA proposes a modelling framework and real time control through model execution via an integrating infrastructure [Klittich 90, CIM-OSA]. To establish the role of the CIM-OSA integrating infrastructure and draw comparisons with the work at LUT a brief description of CIM-OSA modelling and model execution is given next [CIM-OSA, Jorysz 90a,Jorysz 90b,Klittich 90,Kosanke 91,Aquiar 92].

6.2 The CIM-OSA Modelling Framework

Figure 2 shows the complete CIM-OSA modelling framework. This framework has three dimensions which represent:

- levels of generality, these being, generic constructs, partial constructs (i.e. domain generic), and particular constructs of a specific enterprise;
- stages within a system lifecycle, these being, requirements definition, design specification and system implementation;
- modelling views, these being, the functional view, the information view, the resource view and the organisation view.

This modelling framework is proposed for use in supporting the derivation of a CIM system design. The system description model is then released for operation as an executable CIM system model to control enterprise operations in real time. These two phases are described by Kosanke as "enterprise engineering" and "enterprise operation" [Kosanke 91]. Although both

of these activities are described as being supported by the integrating infrastructure, as shown in Figure 3, it is enterprise operation or model execution that has most in common with the work at LUT.

FIGURE 2. .An overview of the CIM-OSA architectural framework

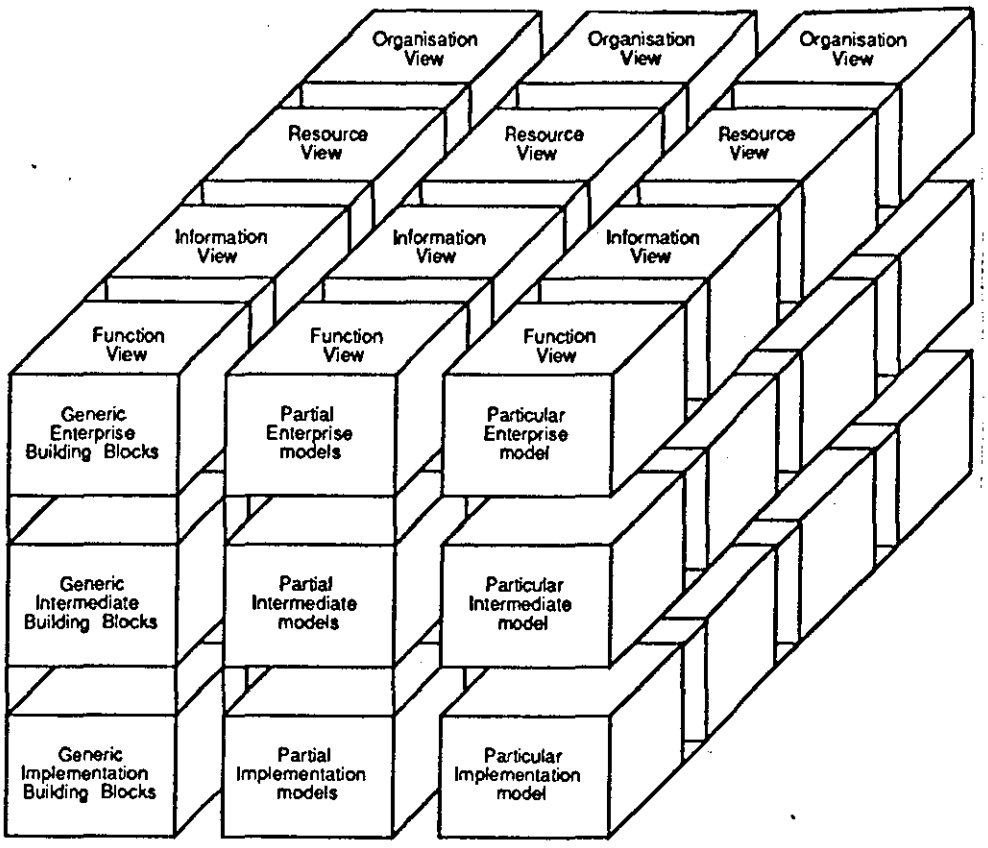


Figure taken from reference: CIM-OSA

6.3 The CIM-OSA Integrating Infrastructure

CIM-OSA proposes an integrating infrastructure which links models to resources (i.e. people, machines, networks, databases) through four sets of services, as follows [Klittich 90,Kosanke 91]:

- Information services, to control and provide access to information;
- front-end services, to connect manufacturing applications;
- communications services, to manage inter-application communication;
- Business services, to provide management of model execution and resources.

The first three of these services are consistent with the information, interaction and communication services identified and implemented in the CIM-BIOSYS integrating infrastructure [Gascoigne 92] (the proposed CIM-OSA front-end services being similar to the interaction services implemented in CIM-BIOSYS). The requirement for CIM-OSA's additional business services is derived from the original top down perspective taken by the CIM-OSA research program. As the function of an enterprise is to achieve business goals, CIM-OSA identifies and models a hierarchical set of business processes. It is these business processes that control model execution through the business services provided by the integrating infrastructure [Klittich 90].

FIGURE 3. CIM-OSA integrated environment showing the Integrating Infrastructure

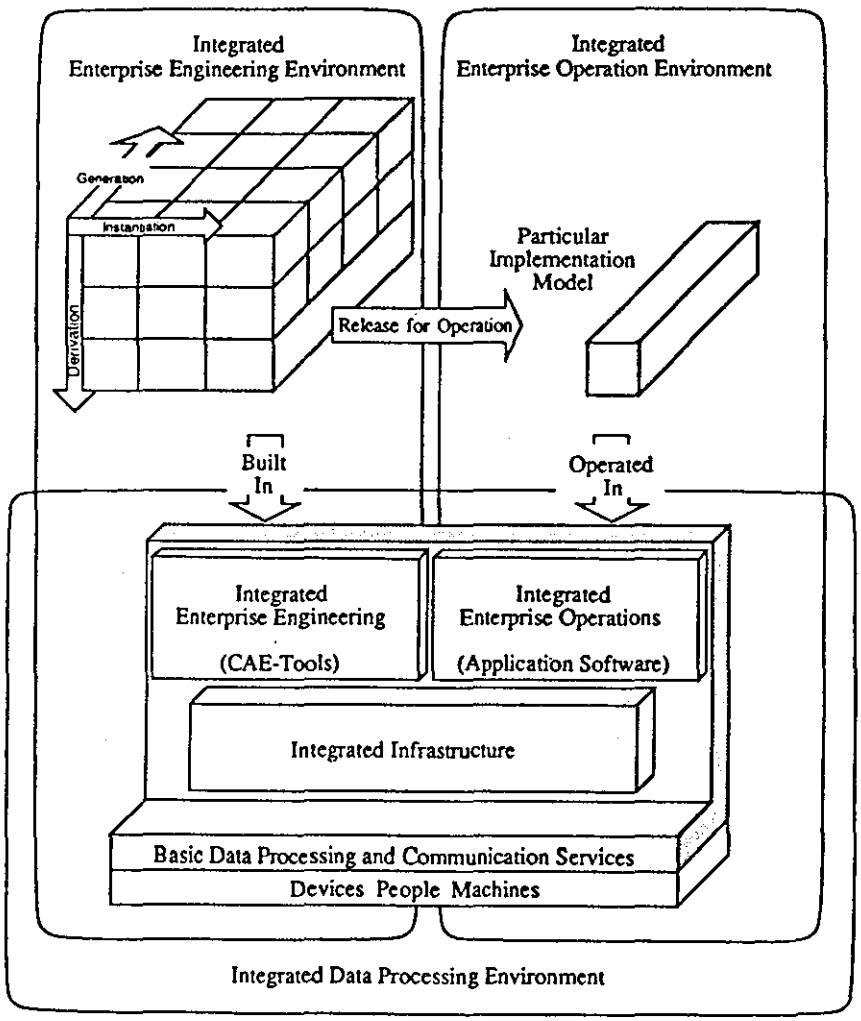


Figure taken from reference: CIM-OSA

With the addition of business services, the CIM-OSA project has concluded the need for a set of managed services similar to those identified, implemented and demonstrated within the SI

group research program at LUT. The following section identifies further examples of proposed or emerging support facilities for implementing open integrated systems.

7.0 FURTHER EXAMPLES OF INTEGRATING INFRASTRUCTURE

During the late 1980's, at about the same time that the role of CIM was being re-evaluated and its scope broadened, related developments were taking place within the field of Information Technology (IT), particularly Open Distributed Processing (ODP) [Brenner 87]. In a response to the widespread establishment of reliable network based computing, the establishment of the ISO/OSI standard, and the rapid increase in distributed processing, ISO launched its ODP standardisation program.

Brenner's paper "Open Distributed Processing" [Brenner 87] details the requirements for distributed processing that was emerging in the late 1980's. Brenner's paper proposes requirements parallel to those which have been identified and implemented at LUT. His description of "distribution transparency" is typical of these parallels. Part of this description is as follows:

- Access transparency: concealing the use of communications when accessing remote resources (programs, data, devices);
- Location transparency: enabling the use of a resource, independent of the placement of that resource in the distributed system;
- Migration transparency: enabling the migration or reconfiguration of resources in a distributed system.

In describing the generic functionality of distributed systems Brenner identifies "supportive services". These, he says, are "a necessary infrastructure of common supportive services to overcome the obstacles inherent in separation".

Brenner identifies a number of research initiatives proposing support mechanisms for ODP (LOCUS [Popek 81], Eden [Black 85]). The following provides other examples.

Support for ODP can be broadly divided into two forms:

- high level language compilers for distributed applications and;
- the provision of additional operating system services for distributed processing.

It is the second form that is closely related to the work at LUT, but examples of both forms are included.

The Conic environment provides a language based approach for building distributed systems [Magee 89]. Conic comprises a set of tools for compilation, configuration, debug and execution within a distributed environment. Of interest are its configuration facilities which provide support for incremental change, and its provision of transparent datatype transformation between heterogeneous processors. These are inline with facilities provided by LUT tools.

Hermes is an experimental language for implementing distributed applications, developed at the IBM T.J.Watson Research Centre [Strom 90]. It offers a language where an application is made up of a set of distributed processes with structured facilities for process interaction.

Nexus is a distributed operating system designed to support object oriented programming in distributed systems [Tripathi 89]. It has much in common with earlier distributed operating systems such as LOCUS [Popek 81] and Eden [Black 85] where the concept of network transparency was demonstrated.

Digital Equipment Corporation (DEC) offer a contemporary solution in their Network Application Support (NAS) product, an open system for applications integration [DEC 92]. NAS comprises a set of high-level services as follows:

- Application access services providing user interface mechanisms;
- Communication and control services for inter-application interaction;
- Information/resource sharing services and;
- System services providing calls to the underlying operating system functionality.

The DEC solution recognises the need for a set of consistent services in line with research at LUT, but it does not provide an underlying infrastructure for system management and configuration.

As is apparent from the above examples, the IT community recognises the need to provide specific mechanisms to support ODP. For those working with CIM, the need to integrate distributed manufacturing application within a highly complex information system has driven the search for an integrating infrastructure and set of consistent services which will underpin the

configuration, execution and maintenance of the system. The following section gives further details of such an integrating infrastructure built by the SI Group at LUT, and used in this thesis.

8.0 DETAILS OF THE CIM-BIOSYS INTEGRATING INFRASTRUCTURE

Figure 4 provides an overview of the functionality of the CIM-BIOSYS integrating infrastructure.

FIGURE 4. A fuctional view of CIM-BIOSYS

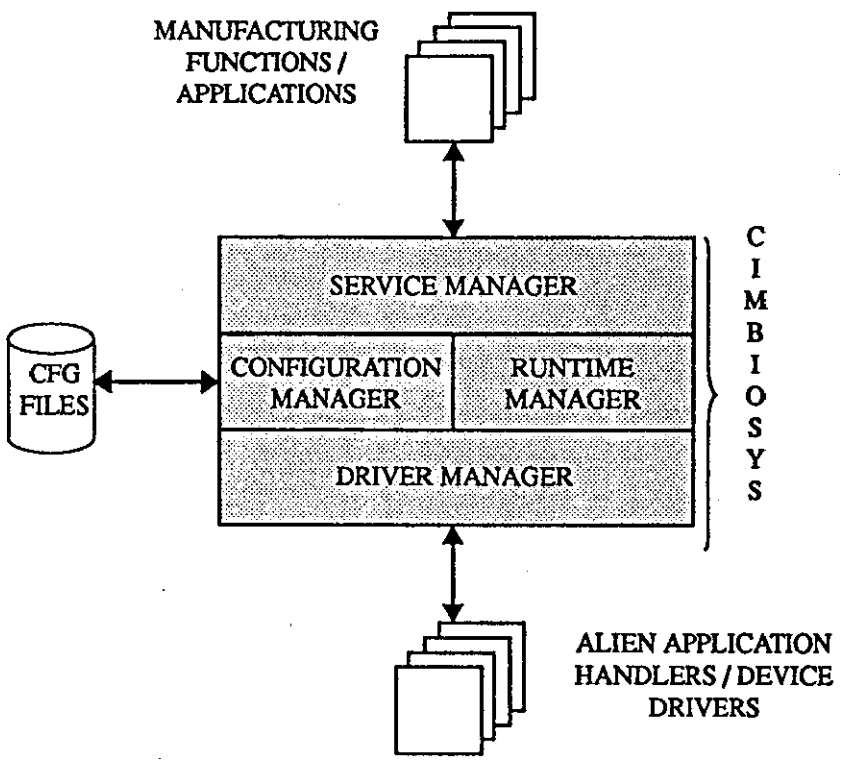


Figure taken from reference Coutts 92

The figure details the four principal functional elements or managers, as follows:

- the Service Manager provides a consistent interaction mechanism for the integration services provided by CIM-BIOSYS to enable application interaction and information access. Typical interaction services include "establish link with another application", "pass data to an application". A typical information service would "open a file in a logical file store";

- the Driver Manager provides similar facilities to the service manager, in the form of a consistent interaction mechanism for device drivers to provide communication between CIM-BIOSYS and remote non compliant devices e.g a robot, an inspection machine, or any device whose processor is unable to support CIM-BIOSYS;
- the Runtime Manager forms part of the infrastructure supporting the integration services. It controls the processes external to and registered with CIM-BIOSYS which use its services. It monitors error conditions within the infrastructure and provides human interface facilities through an Engineer's Interface window. This interface enables full manual control of the Runtime Manager if required;
- the Configuration Manager forms the other part of the underlying infrastructure. The Configuration Manager maintains all internal system configuration data and external configuration files. It also provides CIM-BIOSYS with the configuration facilities that are the key to its functionality [Weston 90]. Human interface facilities are provided through the Administration Interface, which enables examination and manipulation of configuration data.

CIM-BIOSYS is an enabling tool which provides facilities for building next generation CIM systems. These new systems will overcome some of the problems identified in Section 3.0 , in the description of conventional CIM systems.

The integration methodology implicit in the use of CIM-BIOSYS can be further described by considering examples detailing which system components are *not* specified by CIM-BIOSYS, and further identifying and defining what *is* specified by CIM-BIOSYS [Gascoigne 92].

Types of system element not specified are:

- communications media i.e. RS232 via multi twisted pair cable from cell control processor to a remote robot, TCP/IP over an ethernet LAN connecting multiple remote workstations [Pimentel 90];
- application message protocols e.g. MMS [MMS 90a, MMS 90b], application proprietary protocols;
- application conversations i.e. types and number of messages passed between applications;
- degree of complexity of the complete systems.

CIM-BIOSYS does specify the location of the elements within the system which provide the functionality required to enable manufacturing applications to interact. It also specifies how

groups of functional elements will interact. The following definitions of these functional elements are based on the definitions suggested by [Gascoigne 92], Figure 5 presents a pictorial representation describing their inter-relationship:

- CIM-BIOSYS Host Systems - machines/computers/processing platforms which are capable of supporting an instance of the CIM-BIOSYS integrating infrastructure.
- Open CIM-BIOSYS Applications - CIM-BIOSYS compliant applications software written to make use of CIM-BIOSYS services for inter-application interaction and information access;
- Alien Applications - Existing non CIM-BIOSYS compliant applications software having their own proprietary communication/interaction formats. This software will run on a CIM-BIOSYS Host System;
- Alien Applications Handlers - software mainly comprising protocol conversion to turn an Alien Applications communication/interaction format into a CIM-BIOSYS compliant form.
- Alien Devices - machines based on processing platforms which are incapable of supporting CIM-BIOSYS;
- Communication Engines - communications hardware and software essentially comprising the bottom 3 layers of the OSI 7 layer model [Pimentel 90], i.e. the network, data-link and physical layers;
- Communications Drivers - communications software essentially comprising the transport layer (layer 4) of the OSI 7 layer model [Pimentel 90]. Providing access for CIM-BIOSYS to Communication Engine resources to enable the transfer of data packets between CIM-BIOSYS Host Systems. Current implementations are based on inter-process communication (IPC) using UNIX sockets;
- Alien Device Drivers - Communications drivers which provide what ever functionality is required to enable the transfer of CIM-BIOSYS compliant data packets between CIM-BIOSYS host systems (at the OSI transport layer) and Alien Devices. Protocol conversion takes place to enable the Alien Device to interpret the CIM-BIOSYS data packet while Alien Device messages are converted to CIM-BIOSYS compliant data packets.
- Human Interfaces - facilities to enable human interaction with the integrated system, to provide for operation, maintenance, debug and administration. These facilities usually comprise window / mouse and keyboard based systems. The contemporary implementations for use with CIM-BIOSYS support both Sun View and X Windows/Motif;

- CIM-BIOSYS Services - the interaction and information services offered to applications, together with the communication routing facilities which support the operation of these services;
- The CIM-BIOSYS Infrastructure - The runtime management, configuration management, and any other software elements which support, manage or bind together the CIM-BIOSYS services.

This section has described the functional elements of a particular integrating infrastructure, namely CIM-BIOSYS. The following section describes the use of such an infrastructure in underpinning systems integration and enabling the creation of a new generation of CIM systems.

FIGURE 5. The elements of a CIM-BIOSYS integrated system

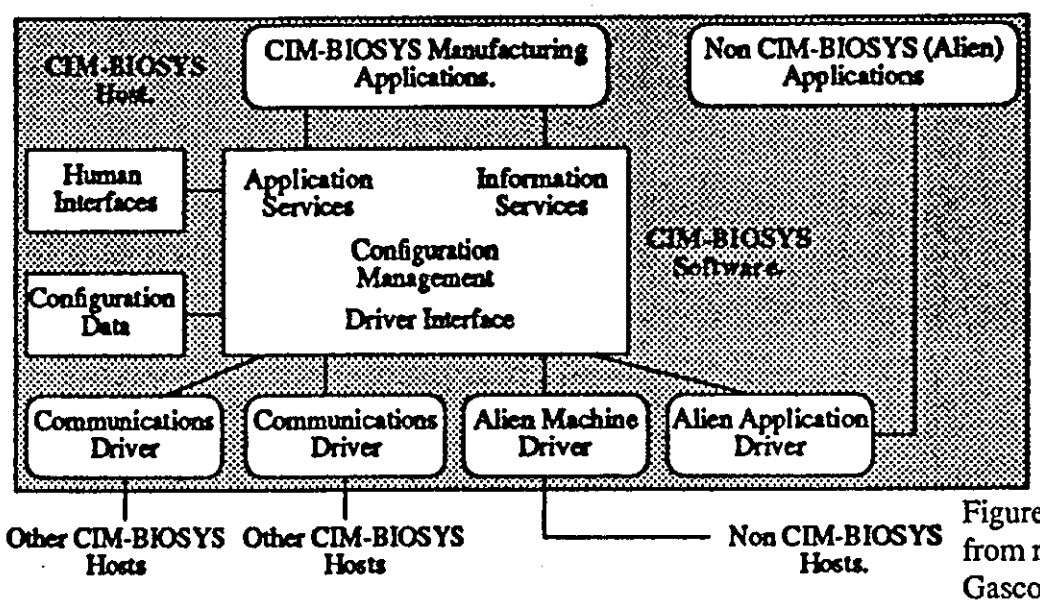


Figure taken from reference: Gascoigne 92

9.0 SOFT INTEGRATED MANUFACTURING SYSTEMS

Section 3.0 of this chapter considered the problems of conventional CIM implementations and pointed up the need for systems that can accommodate change. The ability to accommodate change is required to enable incremental CIM implementation [Ralston 87, Weston 92], and to support the requirements for flexibility within a manufacturing system so that the competitive benefits of CIM can be sustained by an enterprise [Spackman 92].

In order to achieve this required flexibility, it is necessary to identify the knowledge specifically associated with CIM configuration and remove this knowledge from the manufacturing

applications which make up the CIM system. i.e. configuration knowledge associated with application interaction, information access and communication.

In next generation CIM systems all knowledge pertaining to integration issues will be held within the configuration facilities of the integrating infrastructure, while the manufacturing applications require only the knowledge of how to use the integration services offered. Required changes pertaining to the integration issues within a CIM system based on an integrating infrastructure can be met by changes within the infrastructure. The manufacturing applications which make up the CIM system have effectively been decoupled from the integration issues which constrain the ability of the system to handle change.

International research identified within this chapter contends that CIM systems based on integrating infrastructures will form the next generation of more flexible CIM systems. These "soft integrated" solutions will be referred to within this thesis as "*Soft CIM*" systems.

10.0 THE BENEFITS OF SOFT INTEGRATED MANUFACTURING SYSTEMS

There are four principal advantages in using an integrating infrastructure such as CIM-BIOSYS to create *Soft CIM* systems, compared to the contemporary hard integrated systems identified in Section 3.0 [Weston 92, Zhang 92]. These advantages are as follows:

- they inherently deal with complexity;
- they inherently cope with change;
- they provide a methodology for dealing with non-compliant "as is" (legacy) system elements;
- the use of an integrating infrastructure such as CIM-BIOSYS promotes standardisation.

The increased use of integrating infrastructural tools to build CIM systems should encourage the specification of standard applications which could become standard modular building elements (building blocks) of soft CIM systems. The work reported within this thesis covers work by the author in identifying and demonstrating the requirements for the design and building of machine vision systems which can become building blocks of soft CIM systems.

Chapter 4

CONTEMPORARY PRACTICE IN MACHINE VISION DESIGN AND IMPLEMENTATION

1.0 INTRODUCTION

This chapter reviews contemporary practice in machine vision design and implementation. The first few sections of the chapter review the multi-disciplinary nature of computer vision, and contrast academic vision research with the highly structured application of machine vision in industry. The author's interest is in general purpose vision machines suitable for industrial machine vision applications. The hardware and software which make up typical commercial systems is described.

Some of the requirements of next generation machine vision systems are identified. These conclusions are drawn from the author's early vision work together with surveyed literature, and through the discussion with implementers and users of vision technology within industry.

2.0 COMPUTER VISION

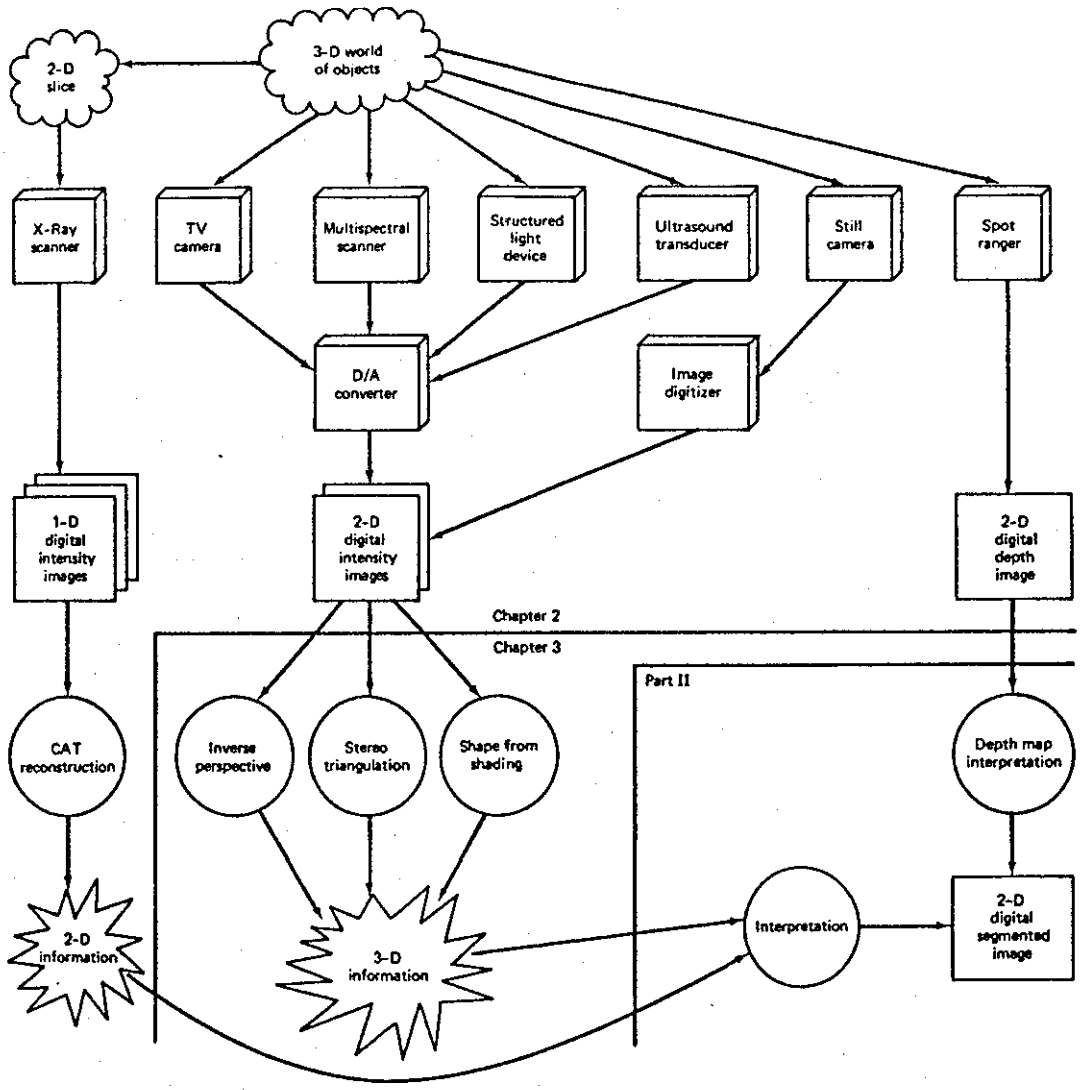
"Computer vision is the construction of explicit, meaningful descriptions of physical objects from images" where these "descriptions are a prerequisite for recognising, manipulating and thinking about objects" [Ballard 82]. Ballard also describes computer vision as the "enterprise of automating and integrating a wide range of processes and representations used for vision perception". Computer vision draws upon a wide variety of disciplines in order to meet the demands of the wide scope of the above definitions. The hardware and software within digital computing, electronic engineering, robotics, and artificial intelligence all contribute to the development of computer vision. Figure 1 illustrates Ballard's view of the scope of computer vision and shows the interaction of the imaging devices, information structures and processes involved.

The three central disciplines of computer vision can be classified as follows [Thomas 91]:

- **Image processing** [Ballard 82, Gonzalez 77, Fairhurst 88]: the science of producing an improved version of an original image, improvement being determined by the requirements of a particular application. Image processing can be further divided into preprocessing, and segmentation. The former deals with techniques such as noise reduction and enhancement of application specific detail, while the later partitions an image into objects of interest.
- **Pattern Classification** [Ballard 82, Fairhurst 88]: the ability to recognise objects within an image through the classification of features extracted following image processing. Typical features include edges and regions.
- **Scene Analysis** [Ballard 82]: the transformation of simple image features into abstract descriptions relating to more complex objects in the scene. This discipline embraces the very complex problem of relating the three dimensional objects within the real world to the two dimensional images available within computer vision. This process is clearly a cognitive one [Thomas 91, Brady 92, Ballard 82] and it is here that the disciplines of image understanding and artificial intelligence make their contribution to computer vision.

Thomas suggests that the central issue of computer vision is "the development of a symbolic scene description from images taken of that scene, in order that the appropriate interaction with the scene may take place" [Thomas 91, Horn 79]. This suggestion is clearly appropriate for visually controlled robotic guidance where sensing, perception, cognition and action form the elements for closed loop position control [OU 92]. Here all three disciplines within computer vision take place: image processing, pattern classification and scene analysis. But, as Thomas also states, approximately 90% of all industrial vision systems are used for inspection. In inspection applications, the central issue is the accurate measurement or reliable classification of particular objects in the scene, where the position and nature of these objects can often be controlled or predicted [Batchelor 85]. Within these inspection applications, sensing and perception are followed by inspection decision criteria. In this field image processing and pattern classification are the required computational disciplines.

FIGURE 1. Related disciplines within computer vision



Imaging devices (boxes), information structures (rectangles), and processes (circles)
Figure taken from ref. Ballard 82.

The application of computer vision by industry has parallels with the application of robotics. Currently they are both applied within very structured environments. In the solution to particular problems which benefit from automated vision, simple and robust automated vision functions have been used. Benefit is gained through increased quality, continuous production or increased flexibility and reduced tooling costs. Chapter 1 of this thesis has identified examples of these "engineered solutions" within the electronics manufacturing industry.

It is because computer vision is a relatively young discipline [Ballard 82, Thomas 91] that its application in industry is taking place in parallel with significant advances in research. A particular focus of research is image understanding / scene interpretation aimed at enabling the

operation of automated entities within unstructured environments. For example, [Brady 92] uses automated vision for the guidance of mobile robots where scene structure is computed through the use of stereo matching of corner features and through the use of a structure from motion algorithm. Similarly, [Durrant-Whyte 92] uses time of flight sensors, laser triangulation and stereo vision for three dimensional scene computation. The Artificial Intelligence Vision Research Unit at Sheffield University studies a broad range of initiatives including stereo processing for 3D vision [Pollard 91, Thacker 91, Rycol 91, Dean 91], and the use of neural network techniques to address some of the issues associated with stereo vision [Mayhew 92].

FIGURE 2. Industrial and academic perspectives for computer vision

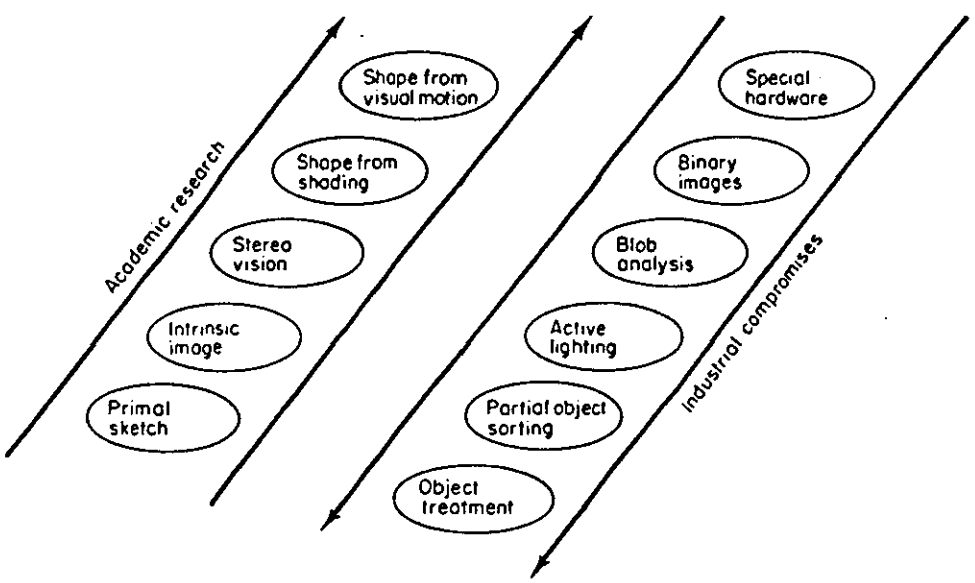


Figure taken from ref. Wright 88.

Figure 2 [Wright 88] contrasts academic research with the compromises made within the industrial application of machine vision. When the products of contemporary research are sufficiently robust to be effectively employed in industry they will provide greater flexibility which could reduce the tooling costs involved in creating highly structured manufacturing environments. However the structured solution to computer vision will remain an economical and robust option, particularly in inspection applications. The work of the author reported within this thesis aims to identify requirements for the next generation of vision applications within structured manufacturing environments. The term "Machine Vision" is often used to describe the application of computer vision within industry. It is the contemporary practice

within the design and implementation of these industrial machine vision systems which forms the main points reported in this chapter.

3.0 MACHINE VISION ARCHITECTURES

3.1 Introduction

Image processing involves the use of a wide range of algorithms for carrying out operations on digital patterns [Ballard 82, Gonzalez 77]. The hardware architecture used to implement these algorithms can take a variety of forms, and are another example of active research within computer vision. At the general purpose end of the spectrum, processing is carried out by software running on a conventional serial digital computer. At the application-specific extreme it is possible to construct an entirely self-contained system composed primarily of dedicated hardware [Fairhurst 88]. In between these extremes there are many hardware / software combinations designed to implement the specific requirements of an application, essentially balancing flexibility and speed.

At the level of image-to-image transformations involving operations on large two dimensional matrices a natural parallelism exists [Nudd 89]. Each pixel or data element within an image matrix is typically treated in an identical manner during a transformation operation. Using a general purpose serial digital computer to handle this type of computation will always be time consuming compared to a machine based on a parallel architecture.

3.2 Parallel Machines

The traditional classification of parallel processing architecture has been in terms of the parallel nature of both the instructions and the data. SIMD (single instruction multiple data) machines have proved to be a popular format. Work at Warwick University exemplifies research in this area [Nudd 92]. The CLIP (Cellular Logic for Image Processing) series of systems are an example of commercially available SIMD machines. CLIP is the product of a research program at University College London [Duff 86].

3.3 Pipeline And Systolic Machines

Pipeline and systolic machines are both further examples of the use of a parallel architecture. Pipelines consist of a set of computational engines arranged in a serial manner which are all processing in parallel. Data is passed from one machine to the next such that the first result takes N units of time to be computed (N is the number of engines in the pipeline). All subsequent results appear after one further unit of time.

A systolic array is a two dimensional pipeline [Wong 85]. An example of this type of architecture has been built at Carnegie Mellon University [Annaratone 86] in the United States.

3.4 Pyramid Machines

Because of the multi-disciplinary nature of computer vision, other elements within a complete system may not be inherently suited to parallel processing. Feature based pattern classification does not use a pixel based format for its data, rather a list based description using tree or graph format is more appropriate. The degree of parallelism is less obvious and some researchers claim much of the processing is inherently serial [Nudd 89]. It should be noted that others working in this active research field claim parallelism is appropriate throughout a system architecture.

Pyramid machines are designed to deal with the requirement for different architecture at different levels within a vision machine. Both the University of Warwick and the University of Massachusetts are building heterogeneous pyramid machines. Pyramid Machines consist of a hardware structure comprising a number of processing layers arranged in a hierarchical fashion. Each layer has a different architecture appropriate to the type of processing done within that layer.

3.5 General Purpose Machines

This thesis considers systems based on the architecture common within general purpose vision machines suitable for industrial machine vision applications. Figure 3 shows the simple overall architecture used within these general purpose systems. The image processing hardware within these systems incorporate various architectural implementations based on some form of parallel processing.

The systems described in Figure 3 comprise vision hardware interfaced to a host processor. The physical interface often takes place through the host system Bus. The host system is used for program development, program control and vision processing tasks which are suited to general purpose serial processing systems. The Imaging board handles two primary functions: image acquisition and storage, and graphical display. The image processing functions take place within the dedicated image processing hardware.

A number of manufacturers offer a range of different architecture within their imaging products [Matrox, Imaging, LSI 93]. Appendix 1 briefly describes two typical examples.

FIGURE 3. Basic elements of a general purpose computer vision system

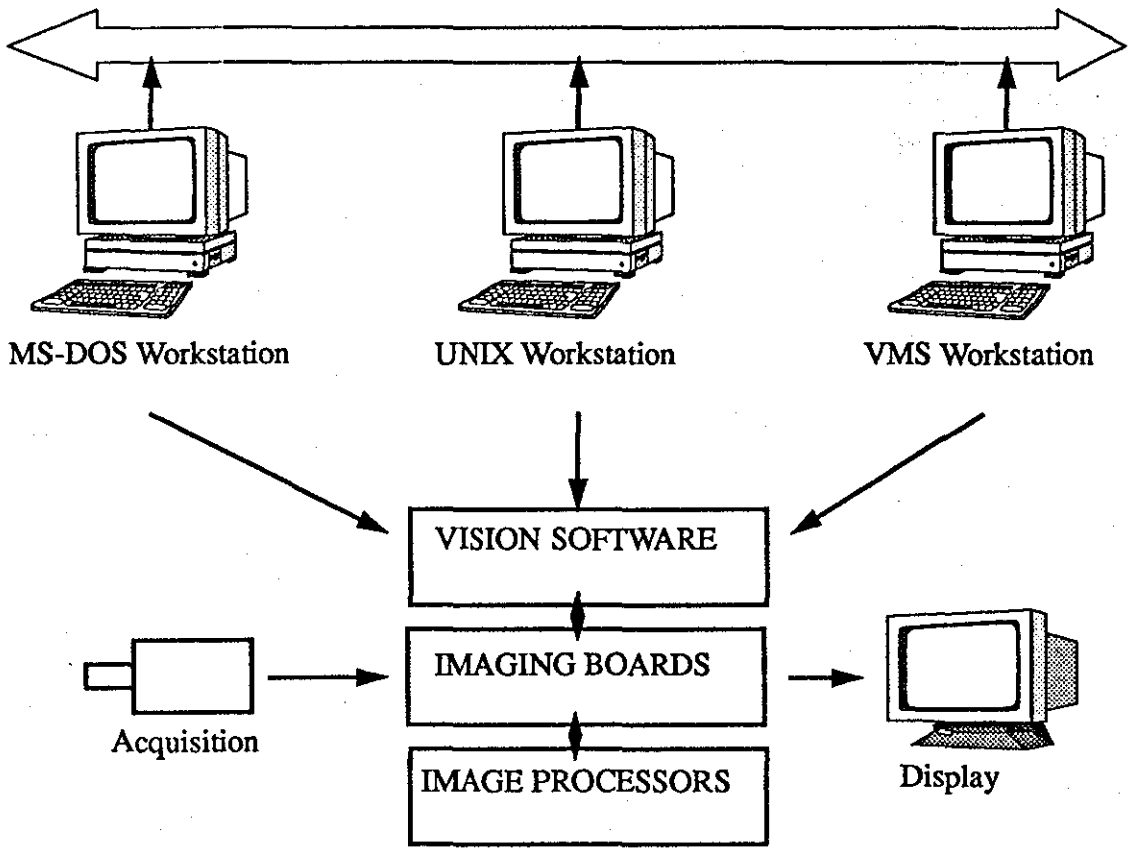


Figure 3 identifies the main software component within the vision machine which controls the operation of the dedicated vision hardware. It is the nature of this software and the processes involved in its programming that are of primary relevance to the work described in this thesis. The following section provides an overview of the nature of this software within contemporary commercial systems.

4.0 CONTEMPORY GENERAL PURPOSE VISION SOFTWARE

4.1 Low-level Function Libraries.

Low-level function libraries are sets of functions which provide access to the imaging hardware. They are often written by the hardware manufacturer and supplied as part of the product. Low-level function libraries are used by vision application developers. They are generally written in C, Pascal or Fortran and provide the programmer with all he/she needs to incorporate the image processing hardware functions into a complete applications program. The application program controls the execution of vision algorithms called via the library functions. The vision library used in this thesis contains functions to drive the image capture, image processing, and the graphics display capability of the hardware [Matrox sw, Town 91].

In order to realise effective vision applications using a low level library, the system developer requires a number of skills. These include:

- general purpose programming experience;
- practical experience of the vision hardware and associated software library;
- an underlying knowledge of the science of image processing to enable the design of vision algorithms.

4.2 General-Purpose Vision Function Libraries

General-purpose vision function libraries are sets of predefined functions that represent many of the more commonly used image processing operations, such as Sobel edge detection [Fu 87]. These libraries may also include functions which implement recognised techniques within particular application domains such as character recognition. They are also offered by hardware independent vendors who offer software which can be ported between particular hardware systems.

general-purpose functions generally fall into the following classes:

- display and image acquisition e.g. lookup table control;
- point to point operations e.g. thresholding;
- mensuration e.g. histogram generation;

- neighbourhood operation e.g filtering;
- frequency operations - Fourier transforms;
- geometric operations e.g rotation;
- morphology e.g. erosion;
- edge detection e.g Sobel.

Typical examples of general-purpose library functions include: thinning passes on a source image, until there is no further change, to provide a skeleton image; and linear filtering to enhance the image.

Considerable skill is still required to realise effective vision applications using library functions at this level. In particular general purpose programming skills and a knowledge of the science of vision processing is required. The use of a general-purpose library can remove the need for knowledge of the vision hardware being used and can ease the required level of understanding regarding the implementation of vision algorithms within the library. This advantage can lead to faster implementation and a lower level of required expertise than is necessary when only low level functions are available.

Vendors of general-purpose libraries often provide an interactive vision processing workbench. These systems enable functions to be called by the operator to examine their effect within a solution to a vision processing problem. The variables associated with each function can normally be adjusted to enable the system developer to understand the requirements for a particular piece of image processing. These systems are available for determining appropriate solutions to machine vision applications but are more commonly used for off line image analysis work.

5.0 BUILDING VISION APPLICATIONS: SOME EXPERIENCES FROM INDUSTRY

5.1 Introduction

The material in this section is derived from interviews between the author and key personnel involved with the operation, application or original design of contemporary machine vision systems. The following section documents the author's work in implementing vision applications using low level and general-purpose library functions. The material in these two sections

help to identify contemporary practice in the design and implementation of machine vision applications and the desires of those who use such systems.

5.2 The Designing of an AOI System For PCB Inspection At Lloyd Doyle Limited

Lloyd Doyle Ltd. design and manufacture a range of Automated Optical Inspection (AOI) equipment for fault finding and verification on PCB artwork, phototooling and innerlayers [Lloyd 93a]. The principal product is the Trackscan 3000 AOI machine which is based on a PCB inspection technique which is unique within the AOI industry. Known as the "Analytical Technique", it uses a method of scanning the PCB and extracting a netlist, describing the tracking on the PCB, which is then compared with reference netlist data. Track width and spacing, missing and spurious copper can be detected by using thinning and fattening algorithms to generate further netlists which are compared with the original. In order to inspect the inner layers of a PCB measuring 18*25 inches with sufficient resolution to identify flaws in track width's of 0.003 inches, very large quantities of data must be processed. Dr. Doyle (Technical Director) quotes typical values of 800 Million pixels processed to inspect a single layer [Lloyd 93b]. In order to achieve this level of image processing in the 2 sq.ft/min typically required within the industry a specialist hardware based architecture is used.

Figure 4 shows a simplified overview of the vision processing hardware architecture which scans the PCB innerlayer and generates a number of netlists. The system is controlled by a host processor running approximately 1.4 Mbytes of Pascal code. The high-level pascal code calls assembler routines which provide low-level control of the vision hardware. The generated netlists are compared with each other and with a reference design netlist derived from a CAD system. This comparison is done by the controlling host software system (which appears as the "CPU compare" block in Figure 4), as is the system configuration and setup which includes the establishment of inspection criteria. The vision processing hardware is special purpose and is inherently inflexible. As in the majority of contemporary products, change has been required at Lloyd Doyle, and these new demands have been incorporated by additional hardware elements and modifications to the controlling software. These modification are undertaken by highly skilled hardware and software designers who are familiar with the Trackscan system.

The Trackscan system can be both a user and generator of information. It requires reference netlist information, preferably derived from CAD information. The physical netlist for individual layers required by the system is not a standard CAD output. There is a requirement to post process information from the current defacto standards which describe the physical layout of each layer of a PCB. Gerber [Martin 90] is a typical example of the defacto standards in use. The Trackscan system generates fault information for use at an off-line rework station and for process management data analysis. Facilities for integrating the Trackscan system within an integrated manufacturing system are more sophisticated than most currently available manufacturing machines in that ethernet communication facilities are available. Information access is via file transfer only.

FIGURE 4. Architecture of the Lloyd Doyle Trackscan 3000 system

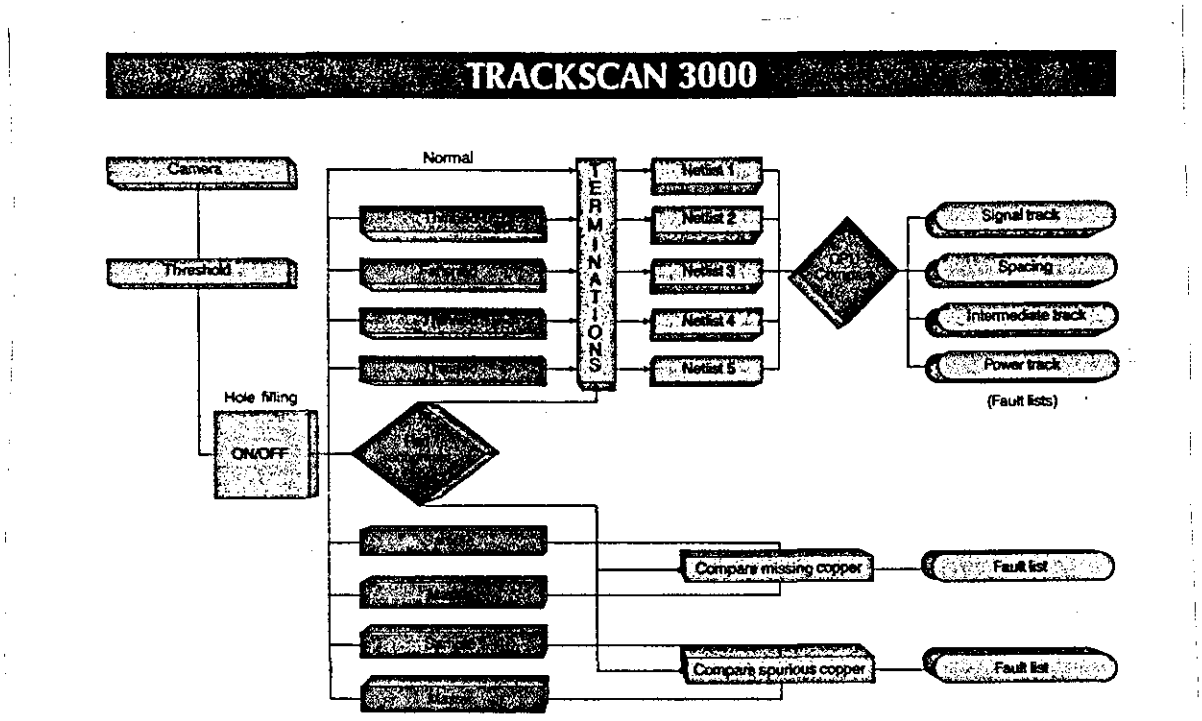


Figure taken from reference Lloyd 93a

The Trackscan system is perfectly capable of operating in a stand alone capacity and many systems sold do exactly this. Reference netlist information can be generated by feature recognition facilities within the system or by generating reference data from a golden board. Fault information can be in the form of VDU display or hard copy. In this form the machine can operate in isolation, configured to solve a specific set of inspection problems. The Managing

Director maintains that incorporating features for user flexibility can put off potential customers [Lloyd 93b]. "Some customers can interpret flexibility as a requirement for increased levels of support and longer setup times", he says. His comment highlights industry's requirement for human integration, which is possible through computer based tool support together with increased levels of operator training.

5.3 Contemporary Experience Of A User Of Automated Inspection During PCB Assembly: Inspection On The SUN Line At ICL Kidsgrove

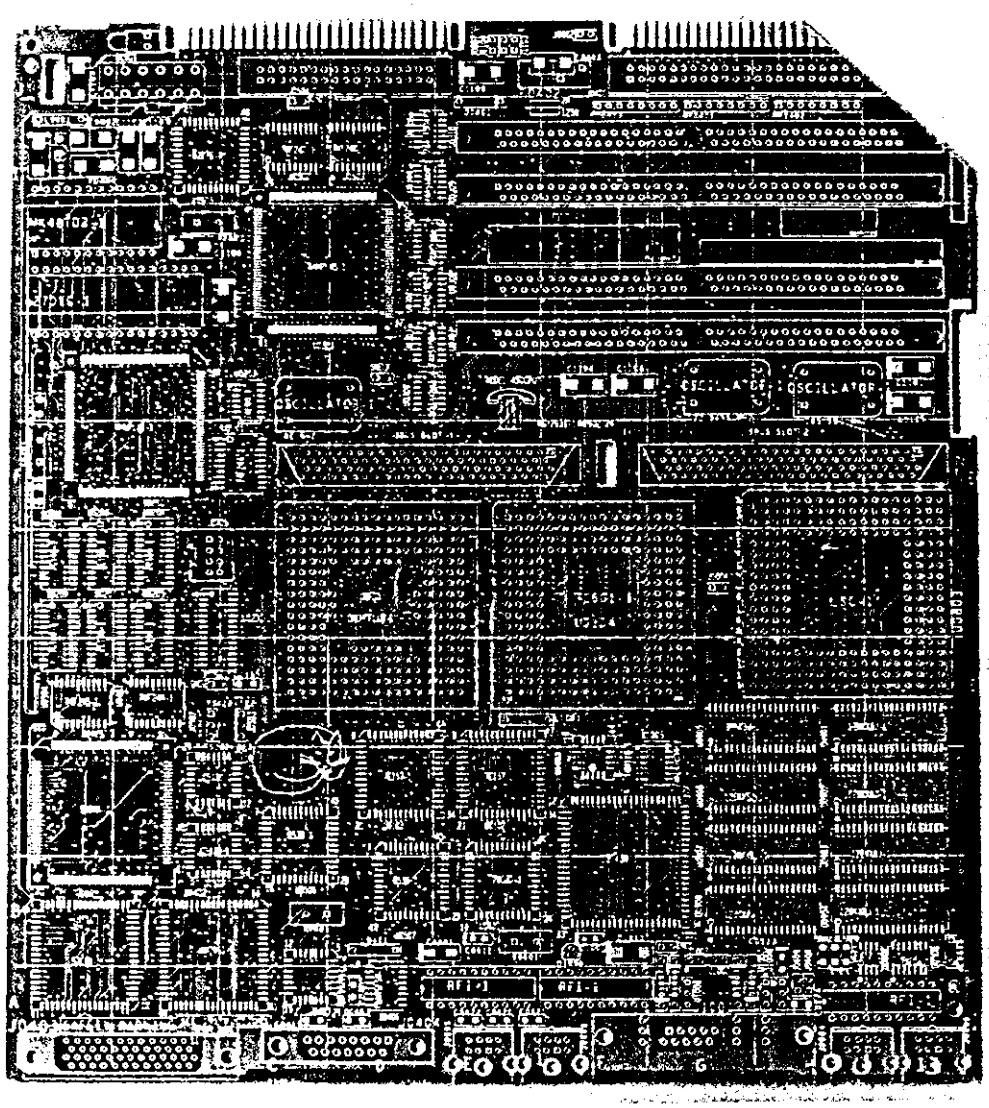
Visual inspection is a key component of the quality assurance procedures which have been put in place over the last decade within the PCB fabrication and assembly process at ICL Kidsgrove [ICL 93]. Human inspection stations using lenses or enhanced/enlarged display systems using colour cameras and high resolution monitors are found at key stages on all the assembly lines at Kidsgrove. The successful use of automated visual inspection of innerlayers during PCB fabrication has taken place at ICL since the mid 1980's. Chapter 1 of this thesis contains details of an automated post solder inspection system which proved to be operationally ineffective during its use in 1989 [ICL 89]. During 1992 ICL took on the subcontract manufacture of two PCB's for SUN Microsystems Inc. On the SUN component assembly line automated optical inspection of soldered boards was once again attempted.

A Control Automation Inc. inspection system was used after the "wash-off" stage which follows wave soldering. The system was to inspect the underside of the PCB's, and was primarily used for identifying missing chip capacitors, misaligned devices and the quality of solder fillets of SMD devices on the underside of the PCB's. The leads of the through hole devices such as Pin Grid Arrays (PGA) and connectors were also inspected for quality of soldered joints. Only two PCB types were ever built on the line with each type having perhaps 3 variants. The machine was situated within the production line flow but was never linked to the shop-floor information system. This was considered unnecessary because of the small number of product variants. The inspection programs detailing the position of each required inspection feature on the PCB, and the associated inspection criteria was stored locally within the machine.

In practice the inspection system required permanent manning. The operator made many adjustments to the inspection software configuration in order to achieve satisfactory inspection performance. One particular problem became apparent during the operation of the machine:

the system was sensitive to changes in the material stock being used to populate the PCB. Typically the SUN PCB's contain 230 components of which approximately 30 are multi-sourced including all those on the bottom side of the PCB. Components from different suppliers were found to be of different shape, size and colour, - characteristics which would alter the performance of the inspection machine. This clearly demonstrates how a management decision, to multisource components, can impact the reliable operation of processes on the shop floor.

FIGURE 5. A PCB which is subject to automated visual inspection

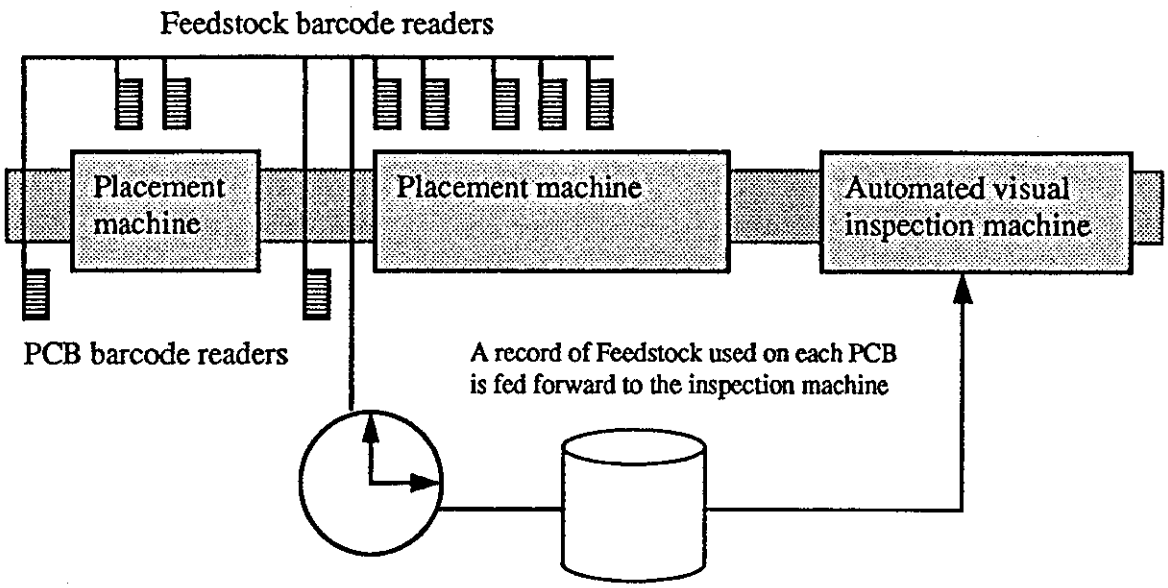


Another problem turned out to be the speed of operation. Figure 5 shows a photograph of one of the PCB's under inspection. The through holes for three one hundred and fifty lead PGA's can be seen, together with four seventy way connectors, two ninety six way connectors, plus further through hole component sites. The solder fillet on each of these sites required inspec-

tion. Following careful set up, the inspection machine could effect the PCB inspection in approximately 3 minutes. The maximum cycle time for the rest of the line was well below this figure.

When the line had reached maturity, i.e. it was well balanced with all its processes under control, the problems at post solder inspection had to be resolved. The solution was to remove the automated system and incorporate human inspection within the "touch up" process.

FIGURE 6. Proposed system for feedstock information system to support automated inspection



Engineers at ICL Kidsgrove contend that reliable automated post solder inspection demands the variables that cause inconsistent operation to be controlled or monitored such that variation can be fed forward to the inspection machine. Figure 6 gives an outline of a system proposed by an ICL engineer to monitor component feedstock on the line. Barcoding is used to identify each type of component, including each supplier variation. Unique barcoding of PCB's already takes place, but it could be used to generate realtime information tying a particular PCB to exactly which component variants were used in its manufacture. This information fed forward to the inspection setup software could be used to configure the ideal inspection parameters for a particular PCB.

The philosophy behind this solution is similar to the successful implementation at NCR in the USA [NCR 89] where inspection criteria were related to specific components. CAD data plus component specific requirements were used to drive the inspection system configuration.

The above scenario implies two principal requirements for future automated final inspection machines:

- they must be fully integrated within a production line information system;
- they require off-line controlling software to effect information management, inspection file configuration, program download and invocation.

Techniques to support these two requirements are demonstrated in the implementation of a soft integrated machine vision system reported in Chapters 9 and 10 of this thesis.

Despite the disappointing conclusion to the two implementations of automated visual inspection during PCB assembly [ICL 89, ICL 93], ICL Kidsgrove recognise the advantages of automating such processes. The intent at ICL is to incorporate the same inspection machine within a new line which will assemble a new generation of highly complex PCB's for Compac. It is expected that this product will have reduced variation in its build and feed stock which could enable effective use of the system. The new line will be high volume, approximately 7.5 to 10 thousand boards per week, and the product manufactured will be high value, approximately 500 pounds per PCB. It is this combination of high volume and high value that drives the need for effective inspection, together with a commitment to SPC. ICL practise Statistical Process Control (SPC) on all their lines and recognise the benefits of automated information generation which could be realised by vision inspection. The present human inspection system relies on the continual collection of data on paper from the inspectors that work the production lines.

6.0 THE AUTHOR'S EARLY WORK IN GENERATING VISION APPLICATIONS

6.1 Introduction

To gain a practical understanding of the requirements for solving inspection problems through the use of digital image processing, I implemented a number of such systems. In conjunction with this work, I have supervised twelve projects in the field of machine vision. This section

gives a brief overview of my early work in generating algorithms for the automated visual inspection of integrated circuits. The methodology and tools employed are typical of those available for contemporary use of low-level and general-purpose vision libraries.

6.2 The Available Tools

The tools available represented the contemporary "state of the art" in their class. They were purchased in 1987 and were of the class of vision system which comprised a PC based imaging board and associated software library.

The hardware available to the author comprised a Matrox MVP-AT imaging board [Matrox pc] mounted in an IBM PS/2 computer system (later updated to an Olivetti 386).

The available image processing software includes both low-level and general-purpose functions. The library is for driving the specific hardware and includes no facilities above the level of frame buffer operation i.e. no feature extraction or measurement algorithms are included.

The Imager-AT library is compiled microsoft C code [Matrox sw]. General purpose software development facilities to utilise the library comprised a Microsoft C compiler with its associated development tools. Tools used by the author include the Quick C editor, The Make tool for controlled compilation and linking, the C Compiler, object code Linker, and the Codeview debugger [Microsoft 87].

6.3 The Design And Implementation Of Structured Application Software

The author's industrial experience has included 12 years in the process and batch manufacturing industry, with responsibility at various levels for the design, implementation, and commissioning of software based control and data acquisition systems. This experience provided an understanding of the requirements for creating structured modular code. General purpose programming languages such as C support structured programming and can be used to create modular code.

In all the vision application work, the author has used existing applications of automated vision within the electronics industry. Existing applications were chosen as the focus of the work is to understand the requirements for designing, implementing and integrating such

applications so that they fulfil the requirements of a building block within a soft CIM system. The implementations use entirely conventional image processing techniques as the author's interests are not in the development of novel vision algorithms.

6.4 IC Inspection During Placement Of SMD Devices

Chapter 2 described a common application of machine vision during the placement of SMD components. The components are picked from a feeder in a random orientation using a vacuum placement head. They are then presented to a vision machine which generates information about the component offset in x, y and θ on the placement head. The vision machine also inspects the component to ensure the correct component has been picked up, and that the component meets a number of quality criteria associated with the component legs.

Three high-level goals were identified for the system:

- get x, y position of the component
- get the orientation of the component
- get leg features from the component

The successful achievement of these goals would enable the generation of the information required by the associated placement machine.

The system was based on a number of contemporary software design paradigms as defined in the following subsections

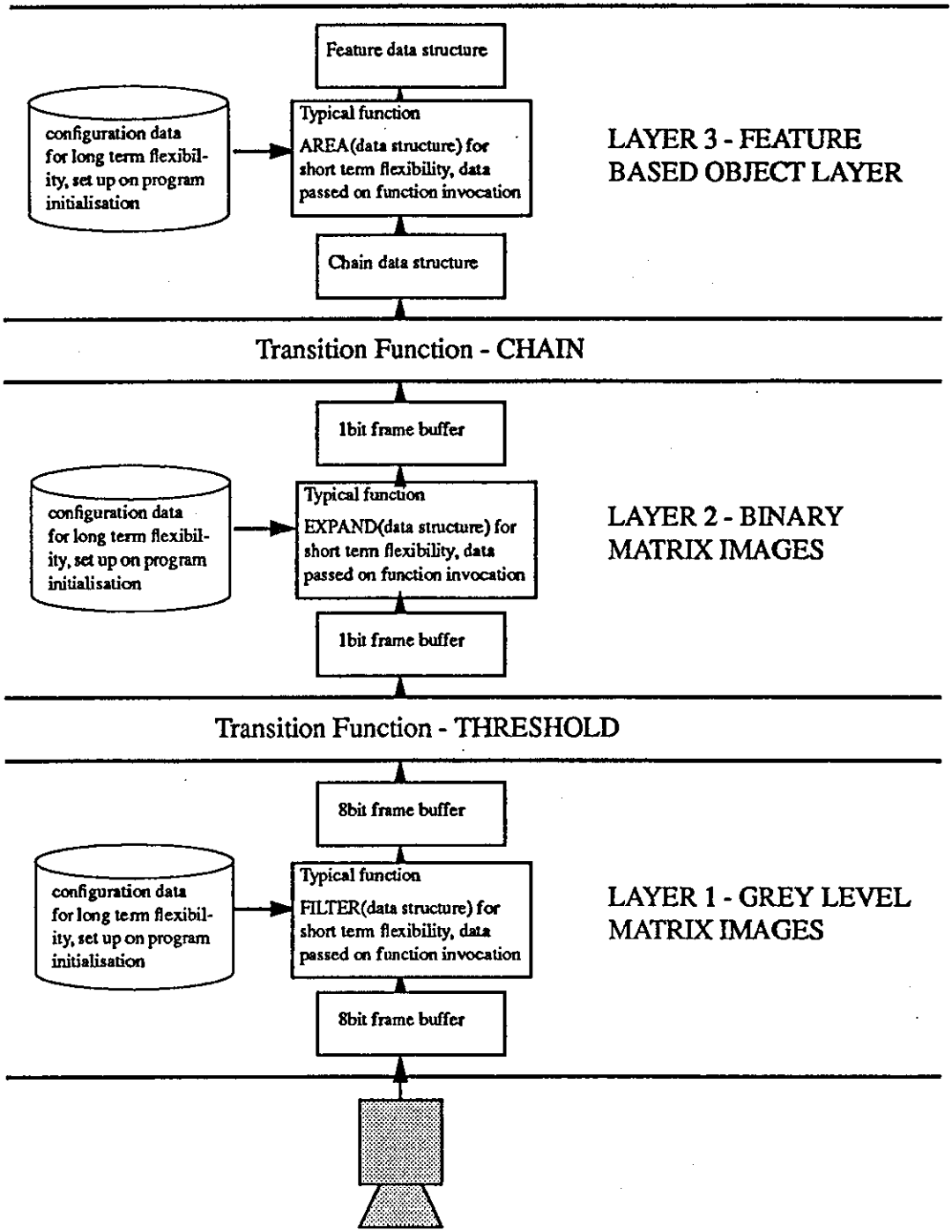
- hierarchical layered structuring;
- data driving;
- modular decomposition.

6.5 The Overall Layered Structure

The system was based loosely on a previous vision system architecture proposed by Azar and Weston [Azar 89]. This comprised a hierarchy of discrete vision layers. The author's decomposition is shown in Figure 7. Particular functions within the architecture are used to bridge the layers and generate specific data structures which form the interface between layers. The

Chain function, an implementation of Freeman Chaincoding [Freeman 74, Pugh 83] is typical of this type of function. Functions within the lower layers typically have a consistent matrix based data structure while those functions in the upper layers have data structures governed by the requirements of the features they describe.

FIGURE 7. Hierarchical decomposition of the author's early vision implementation



6.6 Modular Decomposition

The system makes extensive use of the low-level and general-purpose libraries. The library functions are combined to create modules that are useful within the particular application domain. Domain specific constraints include: for inspection of IC's only one object will ever exist within the scene to be processed.

Typical modules required to implement the high-level goals included:

Layer 1 - greylevel matrix images

GRAB, is used for capturing grey level images;
FILTER, applies a 3*3 convolution to sharpen grey level images;

Layer 1 to 2 intermediate module

THRESHOLD, is used to convert 8 bit grey level images to binary images;

Layer 2 binary matrix images

EXPAND, is used to dilate an object in a binary scene;
CONTRACT, is used to erode an object in a binary scene;
EDGE ENHANCE, implements a sobel edge detection using a 3*3 convolution routine on binary images;

Layer 2 to 3 intermediate module

CHAIN, finds an object within the scene and traces its perimeter using a freeman chain code to describe the perimeter shape.

Layer 3 feature based objects

PERIMETER, computes the perimeter size from a chain code;
AREA, computes an object area from its chain code;
CENTROID, computes an object's centroid from its chaincode;
CORNERS, computes the number and position of an object's corners from its chaincode;
ORIENTATION, computes the orientation of an expected object by comparing its centroid and corner positions with reference data for that object;
VECTORS, computes a vector description of an object from its corner points.

6.7 Data Driven Operation

The software modules were constructed to support long and short term flexibility. Short term configuration facilities within the modules were passed as a data structure parameter when the

function was invoked e.g. source and destination frame buffers, as shown in Figure 7. On building a complete system, an additional module is required to load application specific configuration data into data structures which controlled longer term configuration requirements e.g. type of camera format in the image grabbing function. This architecture enabled a degree of long term flexibility within the system. Reconfiguration could be done by controlled modification within a single module of the system.

7.0 CONCLUSIONS OF THE EXPERIENCES FROM INDUSTRY AND THE AUTHOR'S EARLY WORK

The author's experiences, together with those of Lloyd Doyle and ICL, indicate the following:

- Software design and implementation is practised by highly skilled engineers with very few enabling tools. No structured methodology or relevant architectural standards are available to help lend structure to vision software development.
- machine vision systems will increasingly require facilities to enable their integration within enterprise wide information systems. This contention is supported by the user perspective from ICL. The ICL experiences suggest loaded PCB inspection will never be fully realised without full integration and information support for vision machines.

The problems described in the previous two sections are typical of those recognised during the passed decade as inherent within complex software systems. These problems have been labelled "the software crisis" [Cox 87, Meyer 87]. The principal problems include:

- a requirement for software re-use to speed application development;
- mechanisms to ease modification and change.

It has been suggested that a structured or systemised approach to vision system realisation could contribute to the solution of these problems [Azar 89, Thomas 91]. The following section discusses the notion of systemising vision system realisation.

8.0 MODELS AND METHODOLOGIES TO PROVIDE A STRUCTURED APPROACH TO MACHINE VISION DESIGN AND IMPLEMENTATION

8.1 A Systematic Approach To Machine Vision Realisation

A typical systems approach to complex problems is to decompose the problem and form a hierarchical or a distributed system composed of interrelated sub-modules [Thomas 91]. Thomas suggests that each module should be designed with the system in mind, providing each module with a defined communications interface with surrounding modules. This interface should ensure that detail about the overall goals of the system is propagated throughout the system. This notion is similar to that of Azar and Weston [Azar 88] and of Messina [Messina 91]. What is proposed within the work referenced above is the necessity for a generic modular structure to provide a consistent framework for vision systems. Thomas advocates a domain specific structure with a complete range of subsystem modules sufficient to provide the functions essential to solving problems in the domain. Similarly in this thesis, an application specific structure is developed for vision machines in the PCB manufacturing industry.

FIGURE 8. A generic model for computer vision (The "Classic" breakdown)

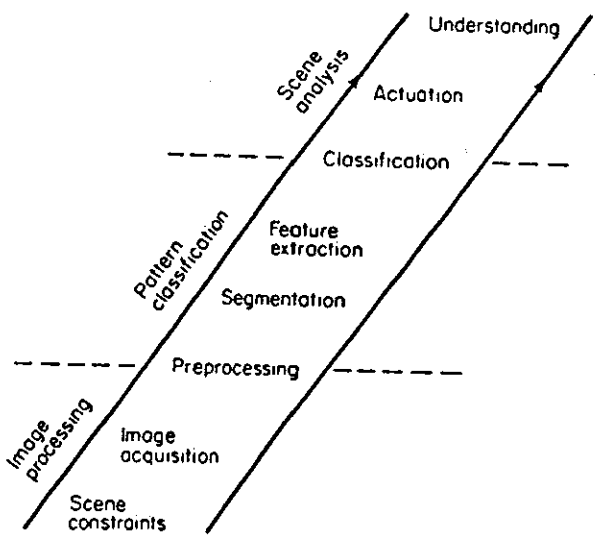


Figure taken from reference Thomas 91

Figure 8 describes a generic model for computer vision, suggested by Thomas, which is in line with a "classic" breakdown of computer vision. This model again identifies the scale and

multi-disciplinary nature of the subject, ranging from optics and the design of structured lighting to artificial intelligence. The three broad disciplines introduced in this chapter underlie the model while modular decomposition within each discipline is identified. Subsystem boundaries exist, but so too does modular interdependence, where typically image preprocessing is dependent on the quality of image acquisition. The levels of current research within the vision related disciplines, identified earlier in the chapter, ensure a requirement for change within individual components of the model.

8.2 Handling Complexity

Over the past 15 years much has been written on the complexity of systems and the need for structured methods for their analysis, design and implementation [Yourdon 79]. The complexity of computer vision systems has been highlighted in this chapter. The software systems which implement complex inspection processes are typical of the "industrial strength software" discussed by Booch [Booch 91]. The future integration of vision systems as part of a manufacturing system will involve even greater complexity. We require mechanisms to cope with this complexity. Three fundamental methods exist: decomposition, abstraction and hierarchy [Cook 91, Booch 91, Seidewitz 86].

Each of these three methods is essential to the systematic approach to vision system creation described in the previous section. Decomposition, the divide and rule principle, is a method for subdividing to create a set of related submodules. These submodules have a defined interface through which they can relate to other submodules. This is achieved through the use of abstraction, the principle of presenting only what is essential at the module interface while hiding all implementation detail within the module.

Seidewittz identifies three useful types of abstraction [Seidewitz 86]:

- Entity Abstraction - an object representing a useful model of a problem domain entity;
- Action Abstraction - an object providing a generalised set of operations which all perform the same kind of function;
- Virtual Machine Abstraction - an object which groups together operations which are all used by some superior level of control or all use some junior level set of operations.

The layers of the classic vision model introduced in the previous section can be considered to be virtual machine abstractions, while the whole model uses a seniority based hierarchy [Dijkstra 68] where the virtual machines provide services to the senior layers.

Two further types of hierarchy can be identified when decomposing complex systems:

- a "part of" or structural hierarchy [Booch 91, Coad 90] referring to decomposition through the different component parts of a system i.e. a car has wheels, an engine, a body shell, etc.;
- a "kind of" or classification hierarchy [Booch 91, Coad 90] referring to decomposition through similar types of a class of entities i.e. the class of vehicle has subtypes car, lorry and van.

The techniques described within this section, which are clearly applicable to the notion of systematic realisation of vision systems, are all key techniques within the object orientated paradigm.

8.3 Object-Orientation

Abstraction is the centrepiece of the object oriented paradigm. An abstract data type within an object oriented program is a model that encompasses a type and an associated set of operations. These operations, commonly termed methods, are defined for, and characterise the behaviour of the data type [Wiener 88].

This combination of state, defined by the abstract data type, and behaviour, defined by the set of associated methods, form a *Class* definition. An *Object* is an instance of a class and a particular application can have any number of such objects. An object encapsulates state by containing a particular set of values for the class abstract data type. The only way to modify the state of the object is by invocation of its methods.

The object oriented paradigm provides "kind of" or classification hierarchies through subclasses. A subclass definition characterises the state and behaviour of a set of objects that inherit some of the characteristics of the parent class but also have their own specialised characteristics [Wiener 88, Booch 91]. A "part of" or structural hierarchy is implemented through the interrelation of a structure of objects [Booch 91]. Figure 9 describes these two orthogonal hierarchies.

FIGURE 9. The relationship between classes and objects within an Object Oriented system

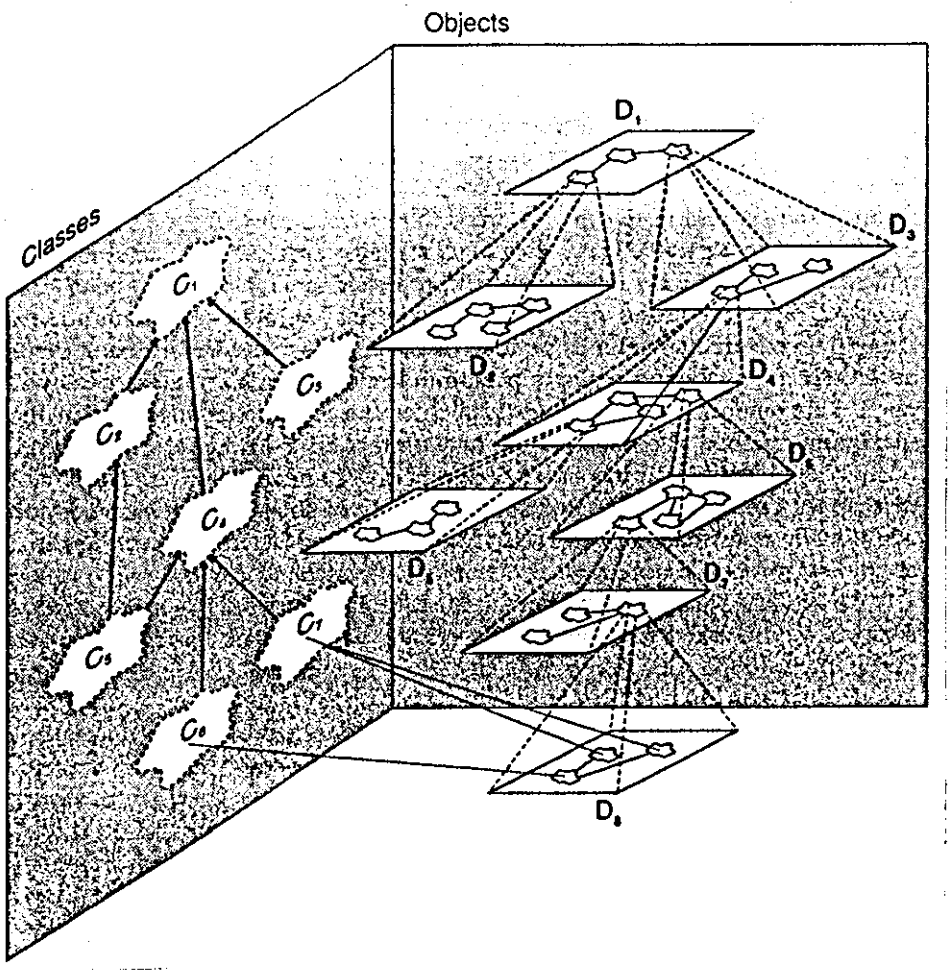


Figure taken from ref. Booch 91

The major goals of object oriented software development are as follows [Wiener 88]:

- to shorten the time and lower the cost of development by using reusable software components in the form of basic classes, and by employing incremental problem solving through the generation of subclasses;
- to lower the cost of software maintenance through the ability to localise changes to the implementation of one or more classes.

Cox makes the same points but introduces an analogy with hardware design through the notion of software IC's [Cox 87]. Within the object oriented paradigm *Encapsulation* provides the mechanism for generating standard general purpose functionality in the form of class

libraries, while *Inheritance* provides a mechanism for customising to produce application specific subclasses.

Object-orientation has been proposed as a methodology which can help to solve some of the problems of the "software crisis" [Cox 87, Meyer 87, Deming 86]. Speed of application development through re-use, and the ability to accommodate change are two principal arguments which are consistent with the requirements of soft integrated machine vision. The requirement for structure through an architectural framework is accommodated by the use of hierarchy in both classification and structural decomposition. Object-orientation is an appropriate paradigm on which to base the design and implementation of a new generation of soft integrated machine vision systems.

Since the work reported in this thesis was begun in the late 1980's several research groups have reported a similar approach. The Philips Forschungs laboratorium in Hamburg have produced a prototype software toolbox allowing image processing algorithms to be implemented independent of computer hardware [Carlson 92]. The system follows a strict object oriented philosophy and is based on a set of data structures and operations from which specific vision applications can be built. Cecchini [Cecchini 90] has reported on an architecture for image processing based on the extensive use of object oriented concepts applied to both image processing and to object representation within a related database. Many more research groups have reported on the suitability of the object orientated paradigm for the design and implementation of integrated manufacturing systems [Rajagopalan 92, Prabhaker 92, Worhach 92, Rogers 92].

PART B

THE DESIGN, IMPLEMENTATION AND MAINTENANCE OF MACHINE VISION APPLICATION OBJECTS

Chapter 5

AN OBJECT-ORIENTED MODEL FOR DIGITAL IMAGE PROCESSING AND FEATURE EXTRACTION

1.0 INTRODUCTION

This chapter describes an object-oriented model which embraces both the state and behaviour of digital image processing and feature extraction. The model is derived using accepted abstractions within computer vision, as defined in both early and contemporary vision literature [Ballard 82, Gonzalez 77, Pugh 83, Batchelor 85, Fu 87, Fairhurst 88, Thomas 91]. The abstractions used are those needed to implement inspection applications within electronics manufacturing. The model is used to explore the notion of supporting design and implementation of machine vision applications with prescriptive structure, using the object-oriented paradigm to provide support for rapid implementation and ease of change. The model is not intended to embrace contemporary research in image processing and feature extraction. The model will form a layer in the complete reference architecture proposed within this thesis.

Use of the model during both design and implementation is considered. Diagramming techniques to support a design methodology using Booch91 notation for C++ implementation is described [Booch 91]. Coad/Yourdon [Coad 90] analysis is used to derive static models, while the Booch91 design notation together with the C++ implementation study yields a richer understanding of the more dynamic relationships between objects in a vision model.

The model can be used to give "shape" or "structure" to the conventional library of proprietary vision functions that are often provided with image processing hardware.

2.0 AN OBJECT-ORIENTED VIEW OF VISION PROCESSING

2.1 Introduction

The systems analysis methodology employed in this thesis is based on that of Coad/Yourdon [Coad 90]. In their book [Coad 90], Coad and Yourdon make recommendations on how to identify objects in the problem space. A definition of the initial problem space encapsulating vision processing used the following relevant potential object sources:

- structure - both classification (generalisation / specialisation structure) and assembly (whole / part structure);
- other systems - systems which will interact with the problem space;
- devices - conceptual devices inside and outside the system;

The primary problem space used to identify the objects within the model comprises the requirement to capture an image of a real 'live' object, process the image and extract and store geometric features which describe the object in the form of an information model representation.

It is assumed in this study that the lighting and camera / lens combination are known and fixed. This allows the work to concentrate on the vision processing aspects of the problem.

2.2 Primary Objects Within A Model Of Vision Processing

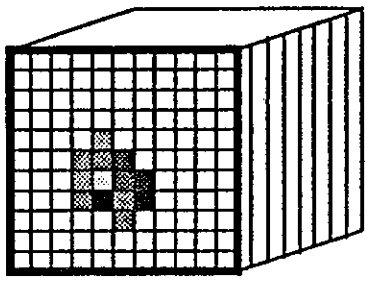
The objects which make up the model are the minimum set required for the thesis work. This set of objects was used to build the first proof-of-concept inspection system and tests the author's notions of model based structure providing support for rapid implementation and ease of change. Additional objects such as descriptor features (typically regions and corners) could be added. This addition could make use of the inheritance mechanism where appropriate. The author's second implementation (described in this thesis) required the addition of corner descriptor features.

These primary objects are based on sets of points as detailed in Figure 1 and described below:

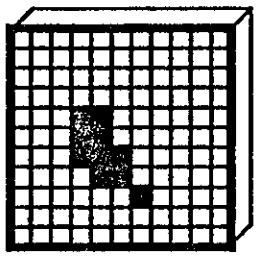
- GREY-LEVEL objects are based on a matrix point set, each point typically using 8 bits of digital information, which facilitates coding of 256 levels of grey;

- BINARY objects are based on a matrix point set, each point using a single bit representing either black or white;
- ARC descriptor objects are based on a chain of points describing the shape of a line connecting two points;
- BOUNDARY descriptor objects are based on a chain of connected points describing a line which starts and finishes at the same point;
- NET descriptor objects are based on connected groups of arc feature objects which form tree structures.

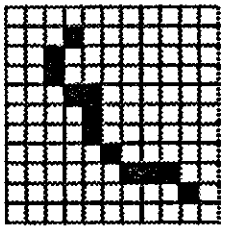
FIGURE 1. Vision objects as pointset descriptions



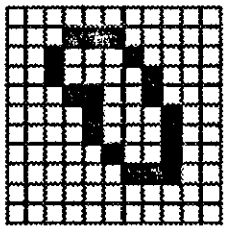
A) grey-level object matrix point set, 8 bits deep



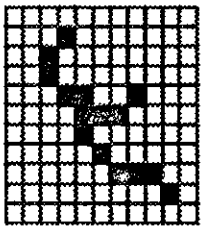
B) binary object matrix point set, 1 bit deep



C) arc descriptor object point set - chaincode describing arc between 2 points



D) boundary descriptor object pointset - chaincode describing a complete boundary



E) net descriptor object pointset - a collection of arc pointsets.

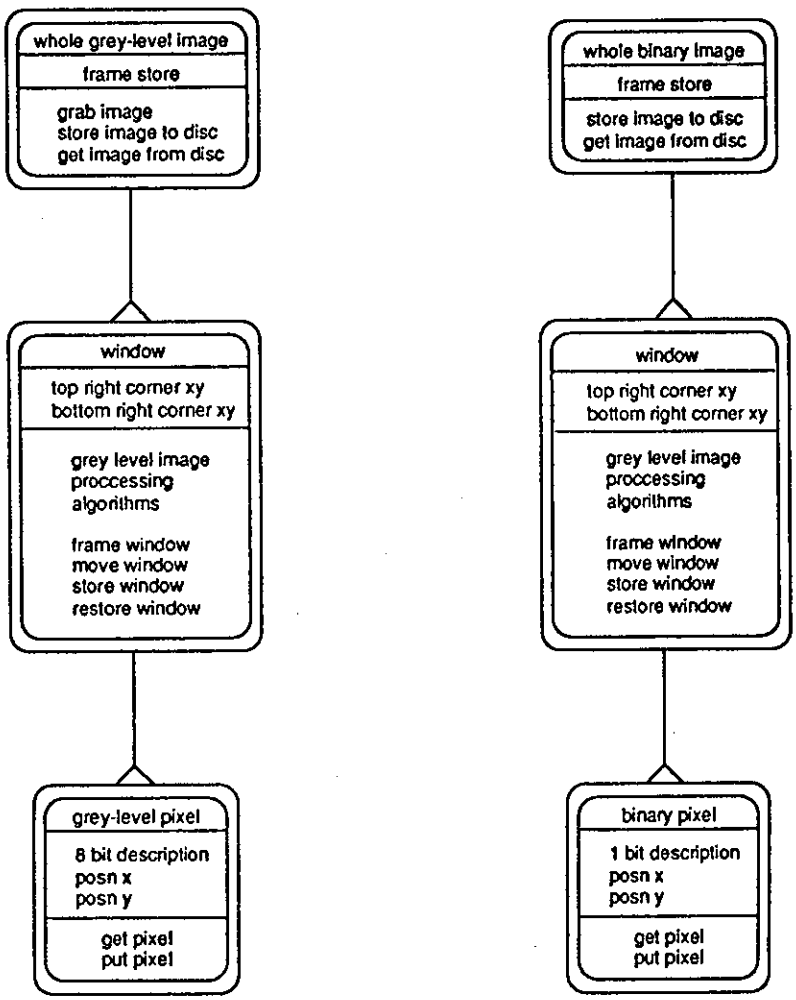
These point sets fall into two distinct categories:

- matrix objects - data abstractions which can be described using a matrix of points. Matrix objects lie within the discipline of image processing, as defined in chapter 3 [Ballard 82, Gonzalez 77, Fairhurst 88]. We assume each pixel is connected to eight neighbourhood pixels;

- descriptor objects - data abstractions which can be described using a set or sets of related points, where these points describe some descriptive feature of a segmented binary matrix object.

These point set objects form the key abstractions of the vision processing domain. The essential functionality of vision processing in this thesis can be defined as the processing algorithms used to perform transforms between matrix objects, and to extract descriptor objects from segmented binary matrix objects. It is these abstractions, their inter-relationships and the transformations which they undergo that form an object-oriented vision model.

FIGURE 2. Matrix level decomposition using assembly structure



The analysis work described in the following section is based on Coad/Youdon [Coad 90] with reference to practical experimentation using the implementation technology described in chapter 4. Some aspects of the model, typically object behaviour, are described in terms of the

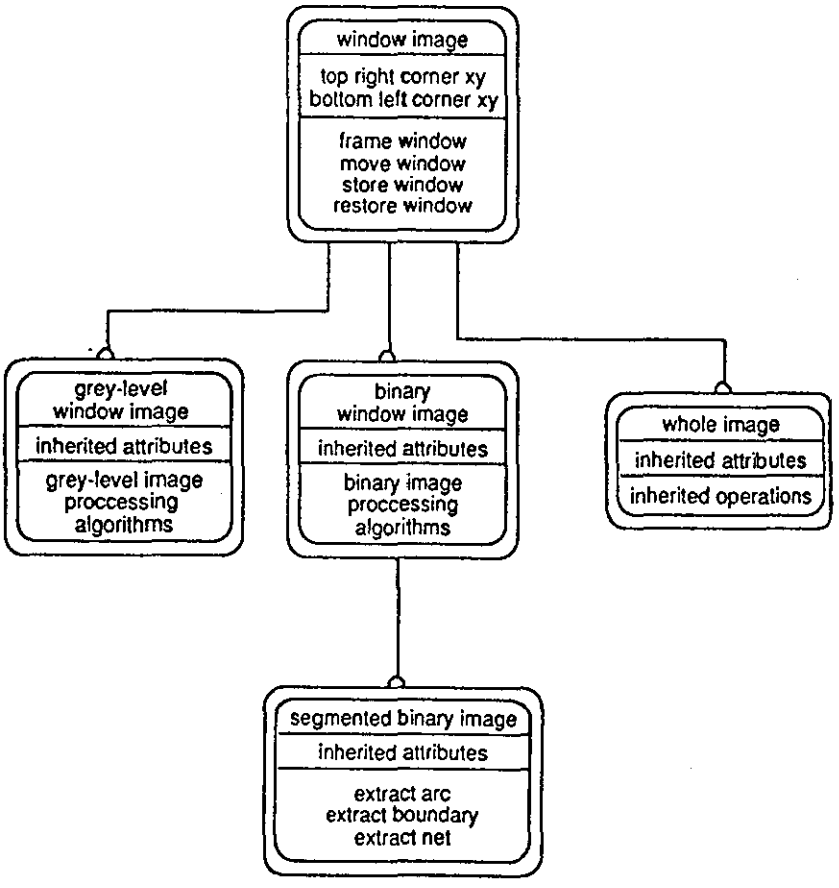
particular implementation. The generic model of vision processing is then extracted from the particular implementation based study. Details of the Coad/Yourdon notation are included in Appendix 2.

2.3 The Matrix Objects - GREY-LEVEL and BINARY

Both grey level and binary matrix point objects can be broken down using assembly structure (as defined in chapter 4) to generate the model in Figure 2. This figure shows a "whole image" which is potentially made up of many "window images" (at a minimum it is made up of a single full size window) which in-turn are made up of many pixel objects. All vision operations, such as "filter" and "edge detect", are included as descriptions of the behaviour of matrix objects, and are implemented as class methods specific to either grey level or binary objects.

By using classification decomposition, the model in Figure 3 was produced. In this model "whole image" is a class of "window image" (having full size window co-ordinates). Grey level and binary windows are also classes of window image.

FIGURE 3. Matrix level decomposition using classification structure



The classification decomposition shown in Figure 3 enables a suitably structured class hierarchy for inheritance to be identified. This class hierarchy can be used when producing an object-oriented implementation model for grey level and binary object classes. The implemented model in the form of an object class library replaces the unstructured set of library subroutines normally offered by the vision hardware vendor. The class library offers a set of related vision objects which abide by generic rules of vision processing.

Figure 3 describes this hierarchy as follows:

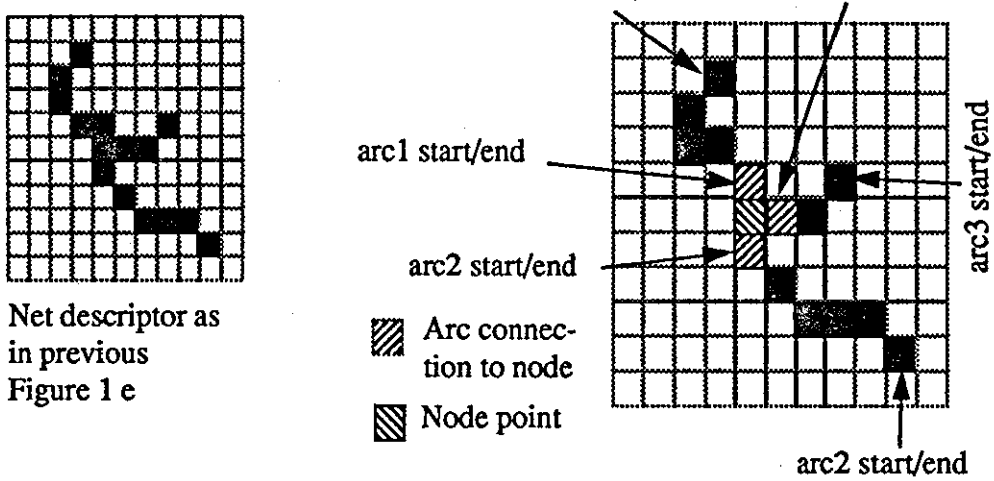
- A base class "window image" encapsulating attributes and methods applicable to all window images. Typical attributes are the name of the frame buffer used to store the image, and the window co-ordinates. Typical methods are "grab image", "save image", "restore image", and "frame window".
- The "grey level" and "binary" classes which are derived from the window image class have no specific attributes but inherit all the attributes of the "window image" class. The important difference between the "grey level" and the "binary" classes is that they have a completely different set of methods specific to either grey level or binary image processing, typically "filter" and "edge enhance" for grey level processing, and in this thesis "erode", and "dilate" are implemented for binary processing. (It is understood that algorithms exist for greylevel erosion and dilation). "Whole image" is a particular class of "window image" which has default conditions for its window coordinates to produce a full screen image, typically used when grabbing raw images.
- The "binary" class has another useful subclass which describes a particular type of binary image termed "segmented binary" objects. This class of object has the same attributes as a binary object, but its class has a further set of relationships with descriptor objects. These relationships are described within the following section. The "segmented binary" object class forms the boundary between matrix and descriptor objects. Its own objects are segmented images from which features can be extracted. These images typically comprise objects made up of thinned single-pixel-width edges.

The decomposition so far has related to matrix level digital image processing. It has identified a set of generic low level vision objects, and the associated methods which can be used in the design and implementation of machine vision applications as library classes to achieve appli-

cation functionality. The following subsection covers the requirement to extract features from the segmented binary objects. Using an early object-oriented analysis technique, we can look for the nouns and verbs in the description of the problem to deduce objects and methods [Abbott 83]. At the descriptor level, features are the primary objects, while extraction is the chief method (the constructor which is used to generate feature objects).

Extracted features, in the form of descriptor objects and their associated methods, can no longer form independent generic object classes which are useful in their own right. This is because the descriptor objects are features extracted for some application specific purpose and this purpose has application specific behaviour which must be embraced by the methods of that descriptor object. We introduce a set of base classes of descriptor objects which can be inherited by application-specific descriptor classes. These base classes contain the description of the feature plus the required extraction methods but do not contain application specific functionality. They are described in the following section. These descriptor classes form the interface between the application layer and the vision model layer within the runtime architecture. This interface is described in the section on the interlayer relationships in chapter 6.

FIGURE 4. The relationship between node objects and arc objects in the matrix space



2.4 Descriptor Objects - ARC, BOUNDARY and NET

The creation and manipulation of objects in a given image processing application enables the generation of a suitable matrix representation of objects of interest within a scene so that feature extraction can take place. What is now required are object classes to facilitate the representation of the objects (within the segmented binary image) that are of interest in such

applications. The methods by which these feature objects can be extracted and manipulated must also be identified.

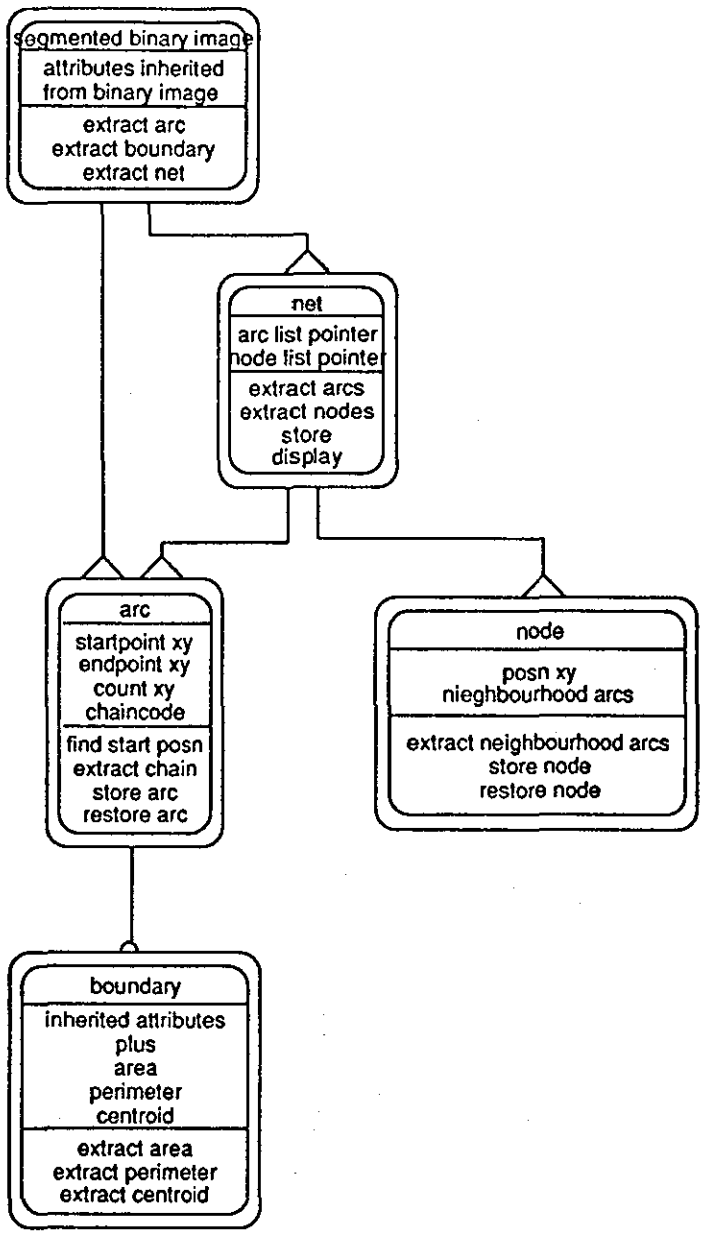
Figure 5 shows the assembly structure of the descriptor objects as parts of a segmented binary object. This is a conceptual view, in line with the statement made earlier, that the descriptor features are a particular view of the segmented binary object. The attributes of a segmented binary object are still those of a set of single bit pixels within a processing window, but through successive image processing transformations it now has properties which can be described by arcs, boundaries and nets. Nets are further decomposed into sets of arcs and nodes required to describe a tree structure. Nodes are the points of intersection of arcs within a net. Nodes are described by their position in the xy matrix space, and by the occurrence of arc end points within their neighbourhood pixels, as detailed in Figure 4.

Figure 5 also shows the result of applying decomposition based on classification structure. It defines the relationship between boundary objects as a particular class of arc objects (i.e. that boundary is the class of arc whose start position is the same as its end position). Having a specific boundary class enables an object to be controlled via a particular set of methods and attributes which are appropriate only to bounded objects i.e. area, centroid etc. and not to the base class of arc.

This approach to decomposition using classification structure is used at the implementation stage to ascertain a suitable class hierarchy. The author's implementation uses arc as a base class, and boundary as a sub class of arc, which inherits and in some cases overloads arc methods.

Section 2.0 has discussed models derived by analysing the image processing domain. Objects, and their relationships within the model, are generic and can be used to form re-usable object class structures for use during the design and implementation of machine vision applications. These objects and their hierarchical relationships could be described as the static aspects of the model. The following section considers the issues involved with the use of these classes and their objects. Further relationships are developed which describe the more dynamic aspects of the model.

FIGURE 5. Descriptor level decomposition using assembly structure and classification structure



3.0 OBJECT RELATIONSHIPS WITHIN THE VISION MODEL LAYER

3.1 The Evolution Of Matrix Objects

When instantiating a grey-level object, an object viewed by a camera is typically snapped by a frame store controller and loaded into a frame buffer memory. The details of such an operation are encapsulated within the grey level class constructor. Binary objects are created by a transformation of a grey level object. They evolve from grey level objects after which the grey

level object is often of no further use. It is this metamorphosis that is not directly embraced by the "class - instance", "inheritance hierarchy", "superclass - subclass" and "meta class" constructs in classical object-oriented methodologies [Cox 87, Booch 91, Wiener 88].

FIGURE 6. The required evolutionary coupling between the grey level and binary classes and objects

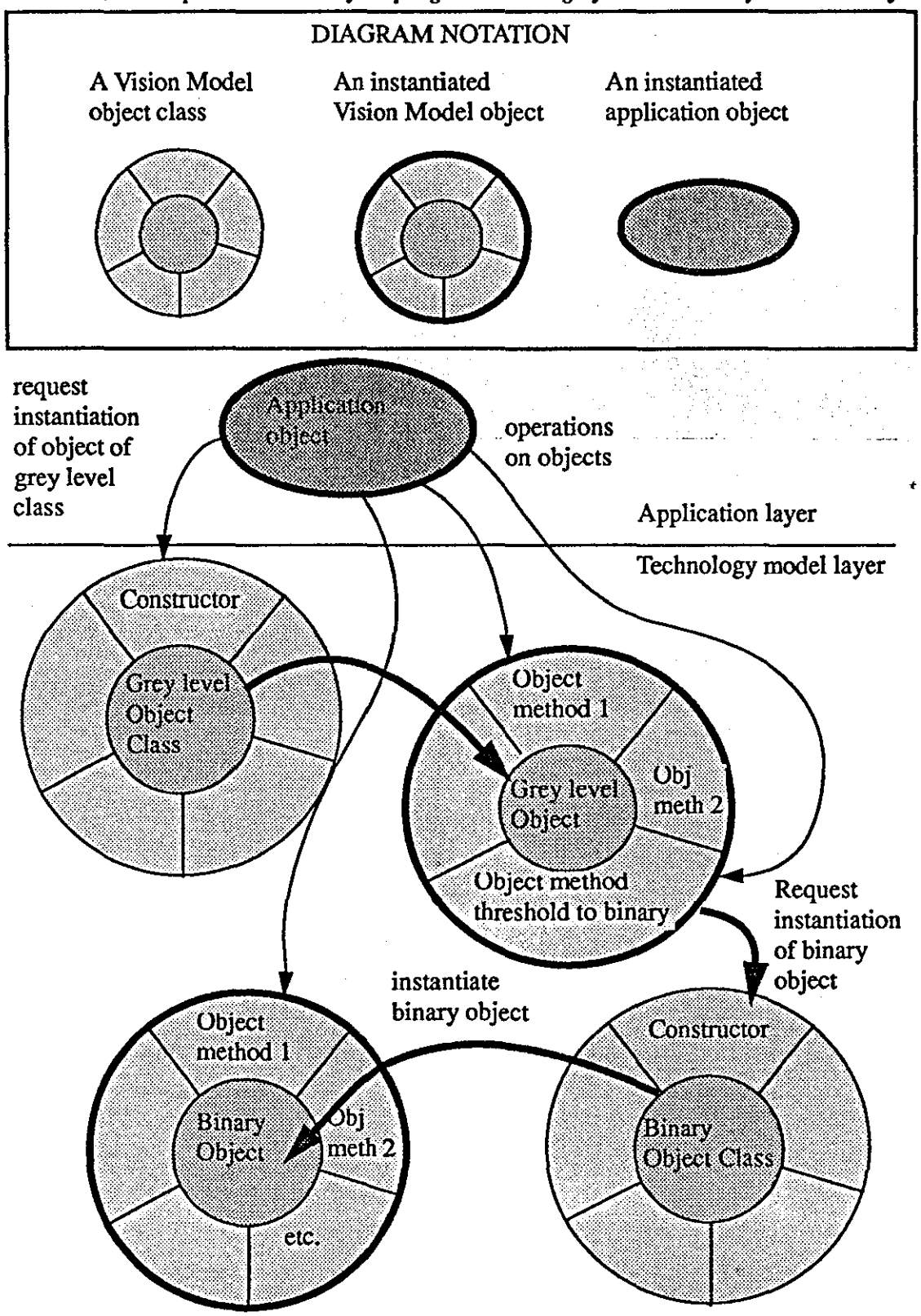


Figure 6 introduces a notation which aims to show the required evolutionary coupling between the grey level and binary classes and objects. The application object will instantiate a grey level object, apply vision processing via grey level object methods, and then apply a threshold method which will create a binary object via the binary class constructor. This scenario complies with object-oriented methods and requires no bending of the rules during implementation, as the threshold method has access to the grey level attributes required to create the binary object. Although the binary object is a different type of object to a grey-level object, it could nevertheless inherit some attributes, such as window co-ordinates and frame store. However it should not inherit the methods specific to the grey-level class. This type of selective evolutionary inheritance would be useful in this domain, but is not supported by the two principal object-oriented programming languages, C++ [Stroustrup 87], and Smalltalk [LaRonde 90].

3.2 Implementation Issues In The Relationship Between Grey-level And Binary Objects

In the C++ implementation, objects of class binary cannot truly be instantiated within a method of the grey-level object class, as the name of the binary object must be passed to the method to enable the application to make use of the object once it exists. Therefore the binary object must be declared and a pointer to it passed to the threshold method which comprises the code implementing the threshold algorithm for the grey-level object. The threshold operation takes place to initialise the binary object using information private to the grey-level object. The binary object now has some initial state while the information hiding principles of object-orientation have been maintained. This implementation is demonstrated in the simple evolutionary object code in Figure 7 showing the code used to implement a simple problem of the evolution of a butterfly object from a caterpillar object.

The code defines three classes, "living_things", "caterpillars" and "butterflies". Caterpillars and butterflies are subclasses inheriting attributes and methods from living_things. The section of code towards the bottom of the right hand column is the main program. This program demonstrates the requirement of the code using the classes to adhere to the evolutionary principle of the model, which is embedded within the classes. The butterfly can only be created by first creating a caterpillar and then calling the caterpillar method "evolve_a_butterfly_called".

FIGURE 7. Code used to implement the evolution of a butterfly object from a caterpillar object

```
#include <stdio.h>
#include <stdlib.h>
#include <stream.hpp>
#include <string.h>
//*****************************************************
//application to demonstrate the problems of a true
//implementation of the evolutionary object instantiation,
//similar to that required within a true vision model.
//The spec for problem is:
//a butterfly air speed is 100 times the caterpillar ground speed
//a butterfly heart rate is 200 times greater than a caterpillars
//only the caterpillar knows its heart rate and ground speed
//*****************************************************
//prototypes
class living_things;
class caterpillars;
class butterflies;

class living_things
{
protected:
int The_heart_rate;
public:
void initialise_living_things(int heart_rate);
void display_your_heart_rate();
};

class butterflies : public living_things
{
private:
int air_speed;
public:
void initialise_butterfly_from_caterpillar
     (int heart_rate,int speed);
void display_your_heart_rate_and_air_speed();
};

class caterpillars : public living_things
{
private:
int ground_speed;
public:
caterpillars(int heart_rate,int speed);
     void display_your_heart_rate_and_ground_speed();
     void evolve_a_butterfly_called(butterflies *a_butterfly);
~caterpillars();
};
//*****************************************************
void living_things :: initialise_living_things(int heart_rate)
{
The_heart_rate = heart_rate;
}
void living_things :: display_your_heart_rate()
{
printf("The heart rate is %d\n",The_heart_rate);
}
//*****************************************************
void butterflies :: initialise_butterfly_from_caterpillar
(int heart_rate,int speed)
{
this -> initialise_living_things(heart_rate);
air_speed = speed;
}
void butterflies :: display_your_heart_rate_and_air_speed()
{
printf("butterfly heart rate is %d and air speed is %d\n"
     ,The_heart_rate,air_speed);
}
//*****************************************************
caterpillars :: caterpillars(int heart_rate,int speed)
{
this -> initialise_living_things(heart_rate);
ground_speed = speed;
}
void caterpillars :: display_your_heart_rate_and_ground_speed()
{
printf("caterpillar heart rate is %d and ground speed is %d\n"
     ,The_heart_rate,ground_speed);
}
void caterpillars :: evolve_a_butterfly_called(butterflies *a_but-
terfly)
{
a_butterfly->initialise_butterfly_from_caterpillar
     (The_heart_rate*200,ground_speed*100);

}
caterpillars :: ~caterpillars()
{
printf("this caterpillar is no more\n");
}
//*****************************************************
void main()
{
caterpillars jim_the_cat(40,20);
//instanciate and initialise a caterpillar using the "caterpillars"
//constructor

jim_the_cat.display_your_heart_rate_and_ground_speed();

butterflies fred_the_but;
//declare a butterfly to create memory space for a butterfly
//(no constructor used)

jim_the_cat.evolve_a_butterfly_called(&fred_the_but);

fred_the_but.display_your_heart_rate_and_air_speed();

exit(0);
}
// rules if this evolution construction is to be used, constructors
//for classes that are to be evolved cannot be used, nor can they
//be used in classes which are to be inherited by a class that is to
//evolve.
```

We argue that this type of C++ implementation represents the most accurate reflection of the natural evolutionary shape of the vision model. This argument is based on our experience gained building C++ vision applications of the type described in chapter 7.

As introduced earlier, binary objects can be further decomposed with the addition of the segmented binary class to describe objects of a suitable matrix representation such that feature extraction can take place. Again, these segmented binary objects evolve from binary objects and cannot be instantiated (declared and initialised) in any other way.

This notion of evolutionary objects breaks down when considering the extraction of descriptor objects. As stated earlier, the vocabulary of the problem leads to the concept of descriptor objects which are instantiated by extracting information from matrix objects of the segmented binary type. This extraction process is described in the following subsection.

3.3 The Extraction Of Descriptor Objects

Although descriptor objects cannot be instantiated without the prior existence of a segmented binary object, they are extracted from, and not generated by, segmented binary objects. This is important, since usable descriptor objects must be application-dependent and cannot therefore become part of a generic vision model. In a C++ implementation, the extraction relationship requires the descriptor object to be a friend of the segmented binary image. This friend relationship allows the descriptor object to access the private data of the segmented binary image, such that the descriptor feature can be extracted. The principles and implementation issues of this extraction interface are detailed in chapter 6.

4.0 DIAGRAMING TECHNIQUES FOR DESIGN USING THE REFERENCE MODEL OF VISION PROCESSING

This section deals with the diagramming tools which can be used during the design of a machine vision application. Such tools are based on diagramming techniques adopted from the BOOCH91 notation [Booch 91]. Primarily, they make use of Class Diagrams and Object Diagrams. The vision processing model is used to provide a richer set of rules to those proposed within Booch's general design methodology.

Class diagrams describe a base structure of classes and of the relationships between classes, while object diagrams describe the use of objects of these classes within an application. A problem solution will yield inheritance hierarchies within the class diagrams of application dependent classes based on the vision model. The object diagrams will describe a complex

combination of both vision model and application specific objects of these classes, typically as shown in the inspection application detailed in chapter 7. It is the underlying model whose structure is always present within a particular solution which provides a constraining framework.

The Booch diagramming techniques used to describe the model are detailed in Appendix 2.

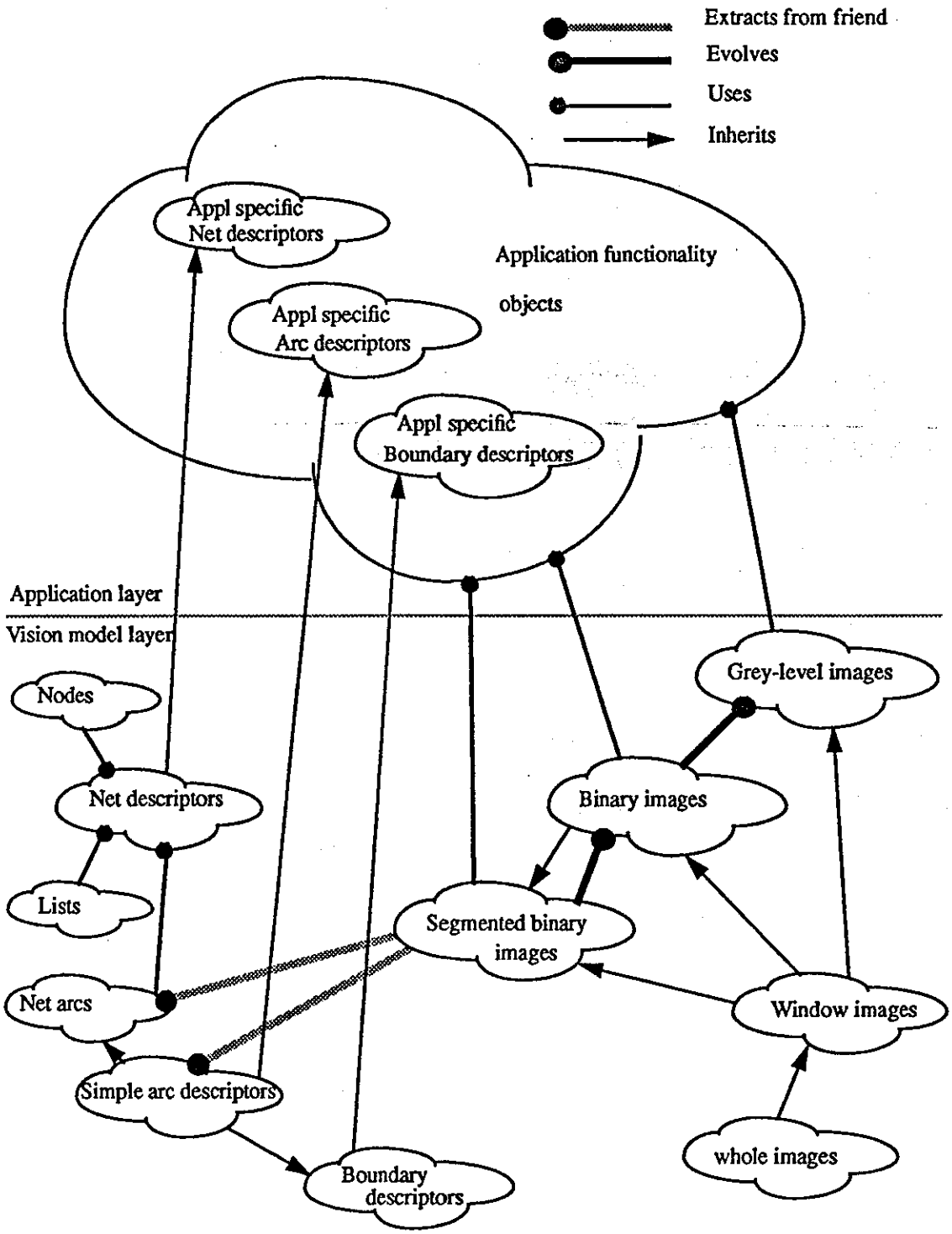
4.1 The Class Diagram

The class diagram describing the vision model and its relationship with the application layer is shown in Figure 8. It describes the principal base object classes for use within a vision system solution and details the relationships between them. The two principal relationships that Booch defines are "uses" and "inherits from". We propose that two additional relationships, derived in the previous section, are necessary to describe the functionality within vision processing applications. These are "evolves from" and "extract from friend". Typically, application functionality "uses" binary images, binary images "evolve from" grey-level images, simple arcs "extract from friend" segmented binary images and grey-level images "inherit from" window images. As with all classes within the model, the net descriptor class contains no application dependent methods. To use the contents of a descriptor level class an application dependent net descriptor class must be created and must inherit the attributes and methods of the net descriptor class. A typical example of this could be a "printed circuit board (PCB) net list description class" made up of a list of nodes and arcs which describe the PCB tracking network. This class might include methods to enable checking of the lists of arcs and nodes extracted from an image against a reference net list to enable the verification of PCBs. One stage of a problem solution would require the development of the class diagram within the application layer to form a set of application dependent classes. These classes would be suitable for producing an interacting set of objects within the object diagram, which would provide a solution to the problem.

The Class diagram provides a graphical description which represents the class structure of the vision model at an abstract level. The use of Booch Class templates provides a means of adding greater substance to the model. The "evolves" and the "extract from friend" relationships

can be added to the template notation. Appendix 2 gives details of the Booch template notation.

FIGURE 8. Vision processing model using adapted BOOCH91 class diagram notation



4.2 The Object Diagram And Timing Diagram

An object diagram and its associated timing diagram are shown in Figure 9 and Figure 10. In these diagrams, a minimal scenario is shown, where one object of each class has been instantiated to describe in more detail the dynamic aspects of the model. In an actual application, the object diagram would become a complex set of diagrams describing the creation, evolution and destruction of many matrix objects, and the creation, extraction, and destruction of many application dependent descriptor objects. Within the object diagram, the containing relationships of the net descriptor objects are explained by the necessary close coupling of the list of nets and the list of nodes.

The timing diagram shown in Figure 10 describes the sequential nature of the image processing and feature extraction operations, and clearly shows the evolutionary creation and destruction of grey and binary objects, while control resides within the application level object. When thresholding to create a binary object, the application object passes control to the grey-level object. The grey level object then creates the binary object and control is passed back to the application object. The application object can now pass messages to the binary object invoking the binary image processing required by the application. A similar process takes place when creating a segmented binary object. Having created a segmented binary object, control is passed back to an application object which can now create an application dependent arc object. The arc object will extract features from the segmented binary object, but never return control to it. The figure describes 2 further application dependent arc objects being created.

This type of timing diagram has been included to present another view of the thread of control within a single image processing and feature extraction operation. In the author's work it was never found necessary to use these timing diagrams to describe application solutions. Booch states that all object diagrams do not require associated timing diagrams, but they can be useful where time critical functionality is to be described [Booch 91]. For example, although the thread of control within an inspection application may well be sequential, timing diagrams can be used to document real-time deadlines.

FIGURE 9. Vision processing model using adapted BOOCH91 object diagram notation

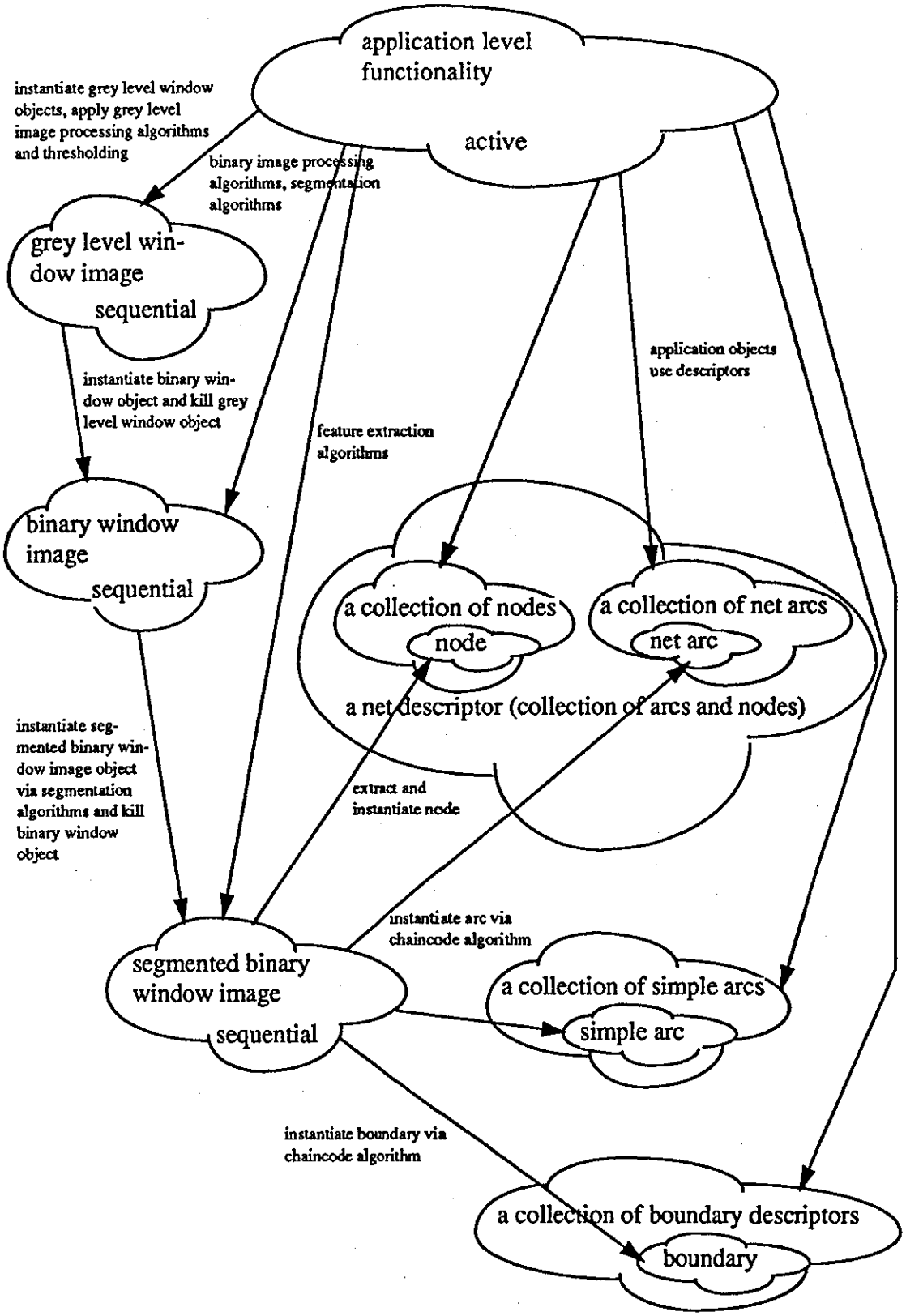
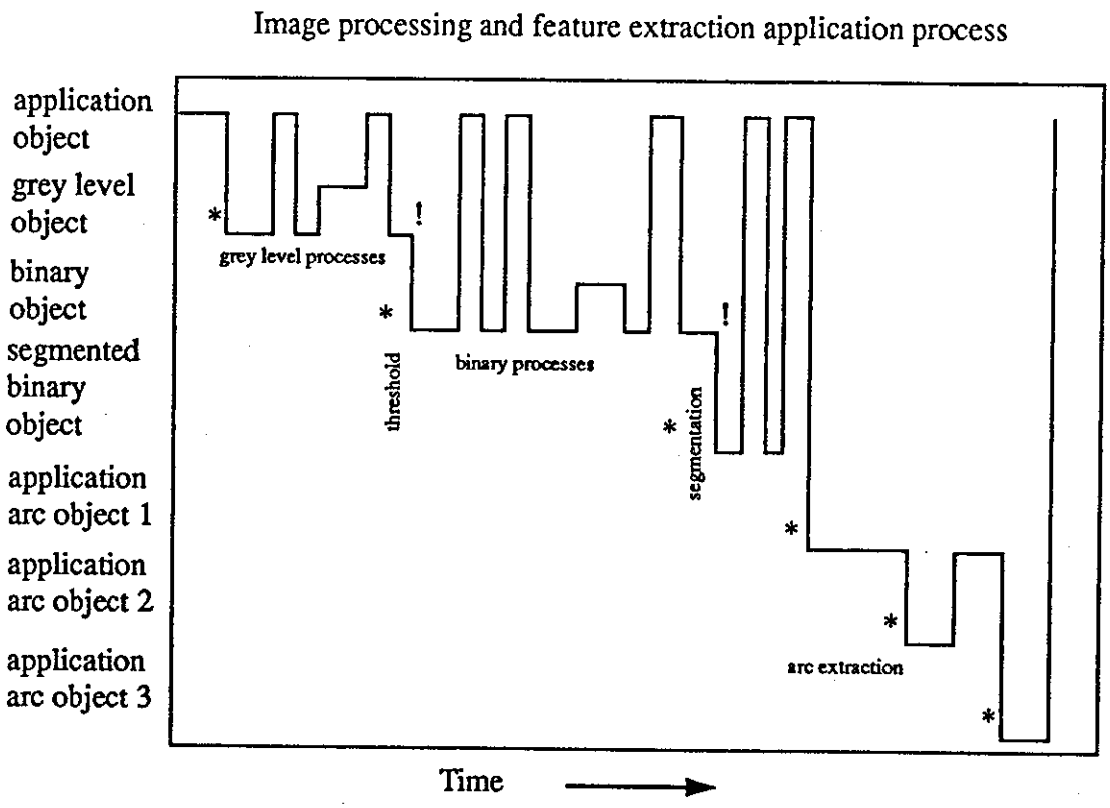


FIGURE 10. Vision processing model using adapted BOOCH91 timing diagram notation



Chapter 6

THE STRUCTURED DESIGN AND IMPLEMENTATION OF VISION APPLICATION SOFTWARE

1.0 INTRODUCTION

The aim of this chapter is to discuss further two issues introduced in chapter 1. These are central to the techniques proposed in this thesis:

- the concept of information model driven machine vision applications, and
- the proposed layered architecture for machine vision applications software.

Information derived from models of the inspection object of interest are used to improve contemporary means of engineering machine vision systems, at both design-time and run-time:

- by providing appropriate structure which formalises vision application software creation, and;
- by characterising properties of the runtime system, to provide the required flexibility so that multiple objects (adhering to the same neutral model) can be processed by the system without the need for reconfiguration or reprogramming.

Section 2.0 considers the role of product models in the design of application software. It proceeds to develop the notion of "model driven application design". The benefits in terms of software modification or ease of change have been identified. Section 3.0 details the architecture proposed for applications software, which combines vision application and vision model issues. The architecture also addresses the need for flexibility in vision hardware implementation. The author's architecture is contrasted with an established model based on a hierarchical stack. Section 4.0 considers interlayer relationships, based on a specific C++ implementation. The proposed interfaces allow layer to layer decomposition within design, and isolation within implementation, but do not constrain application design or choice of vision hardware resource.

2.0 MODEL DRIVEN APPLICATION DESIGN

2.1 Introduction

The application software in a machine vision system provides the mechanisms which implement the application specific functionality of the system. Software can be designed and built to meet a spectrum of requirements, ranging from a specialised user requirement to one which encapsulates a more general need. It is this software which provides the user with the vision services his or her application requires.

Chapter 5 suggested that application software can be made up of application specific object classes. These application specific object classes inherit the generic descriptor classes which encapsulate feature representations and their extraction methods. They use the grey-level and binary matrix classes to provide the required image processing prior to feature extraction of their descriptor objects.

We describe how information models of the principal objects can be used to direct the design process. This is achieved by identifying potential application specific objects and their relationships within existing available information models.

2.2 Models And Model Views

A model is a way of formally describing an entity in terms of its attributes and relationships with other entities, usually in the form of function and information [Yourdon 79]. A single entity may require any number of different models to describe it, depending on the requirements of the system that is to make use of the model [Kosanke 91].

A number of contemporary research initiatives have focused on defining the necessary form of Product Models (such as PDES/STEP [Owen 87, Palfremen 90]). These models will be available within manufacturing information systems of the future and will describe many aspects of a product, to support its lifecycle from design through to implementation and support in the field. Contemporary research proposes that these Product Models should be of a neutral form [PDES 87, STEP 89, Clements 91], from which the various forms of information can be derived to support manufacturing applications throughout the product lifecycle.

Figure 1 shows the concept, used in this thesis, of application dependent views of a neutral product model, This concept enables multiple application specific views of the model to be generated.

FIGURE 1. Application dependent views of a neutral data store

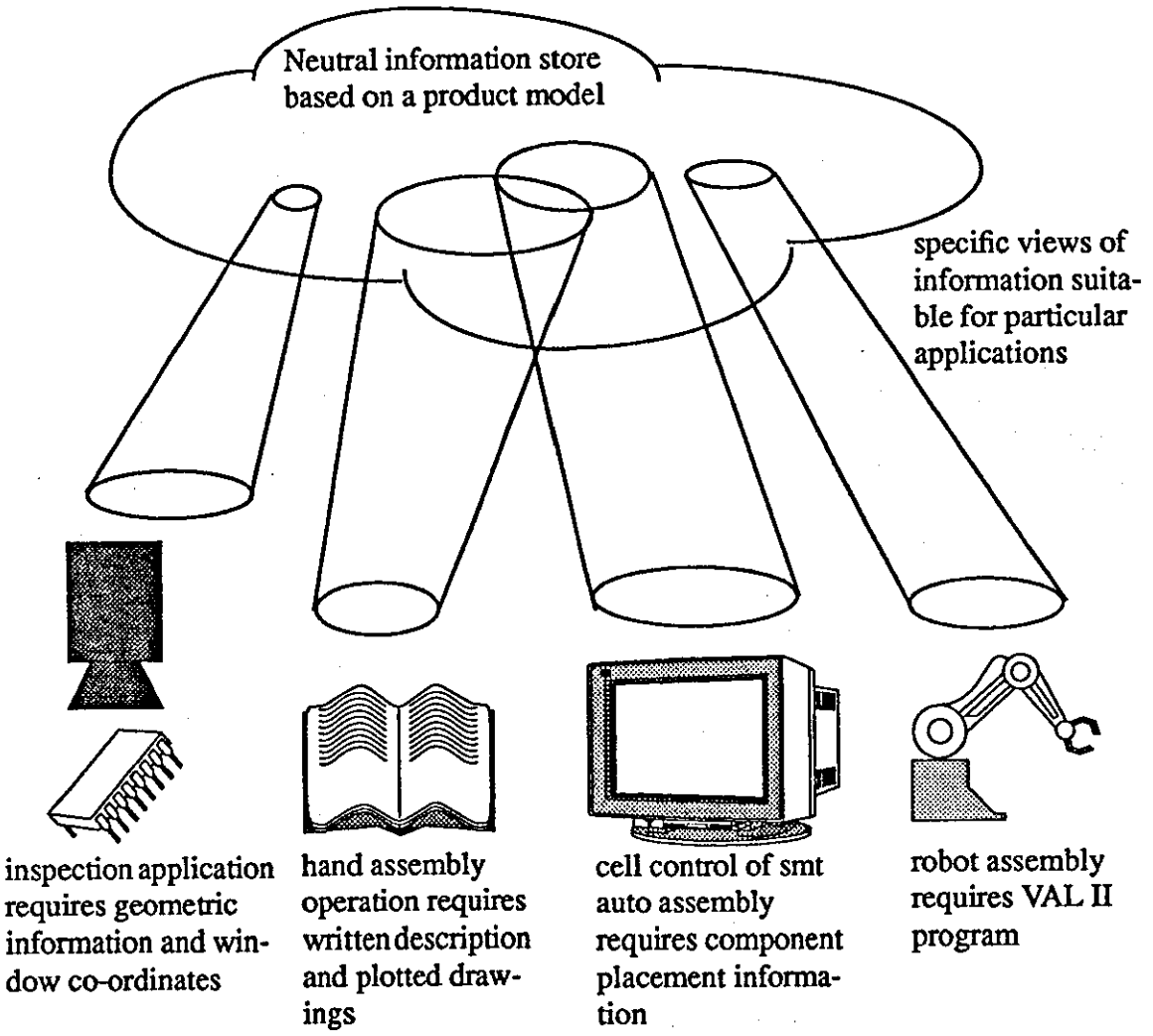
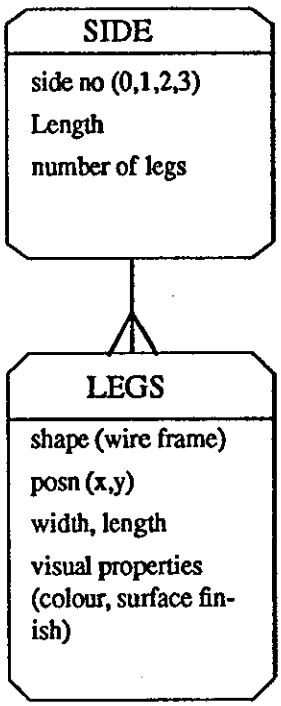


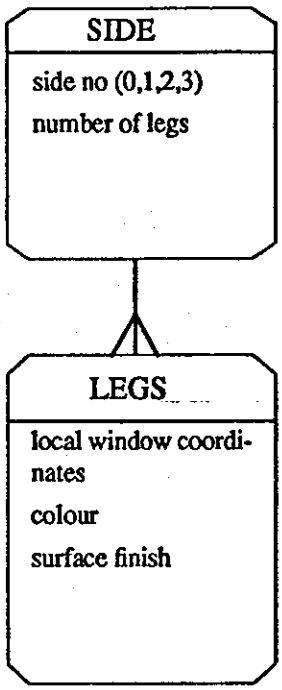
Figure 2 shows how four different model views are required within the design, implementation and execution phases of the lifecycle of a component inspection application. The Product Design Model shown in part A of Figure 2 represents a fragment of a complete design description of the physical attributes of a component. (The form of the model illustrates the model driven concept and is not proposed as a formal model). Parts B, C and D of Figure 2 show examples of application specific views of the model. Part B includes information which defines the physical, spatial relationships between entities of the model, where this model is used during run-time. This is derived from the design model and is passed to the application during run time execution to configure it, thus enabling image processing of a specific compo-

nent (i.e. the blob colour could determine the required threshold value, whereas the leg size and its position will determine the local processing window coordinates). Part C is the model view generated during application design to identify the attributes or methods of the application specific object classes. It is view C that becomes an application developed object class in a Booch Class or Object Diagram.

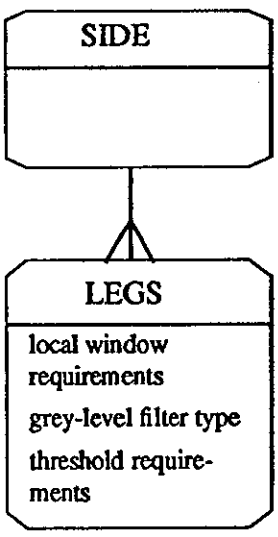
FIGURE 2. 4 views of a partial model for component side and leg entities



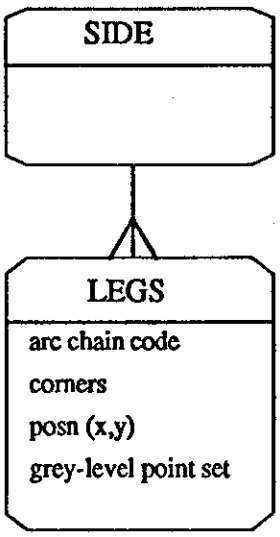
VIEW A - the product design model



VIEW B - local view for driving the vision inspection application



VIEW C - vision application s/w requirements model



VIEW D - vision application generated - live model

Part D describes the form of the "live model". In this work the term live model refers to a representation of an object being inspected by the vision machine. The model is made up of features extracted during image processing and feature extraction from an image of a real object. This model is not a view of the neutral product model.

In order to complete a comparison inspection function, two more model views are required. A model used for comparison would describe more detailed features such as lead pitch, leg dimensions etc. [Fuji 88]. One instance of this model would be a view of the design model, while the other instance would be a view of the live model. Both views would require separate and different view provision software to extract and process the relevant information from the different information formats of the design and live models, and to populate the application specific models for comparison.

This study concentrates on issues relating to the design and building of integrated model driven applications. The models for comparison, associated with the inspection checking operations which are specific to particular application functionality, are not considered.

2.3 The Use Of Objects In The Application Domain To Provide Structure Within The Design Of Application Software

When designing a software solution using principles of object oriented design, the objects identified within a particular application domain will make up the major objects within the application software [Coad 90, Booch 91]. A particular machine vision inspection application which forms an integral part of a Soft Integrated Manufacturing System should be supported during its design phase by appropriate Product Design Models of the objects to be inspected. We propose that the objects and sub-objects, which comprise the supporting Product Design Models, also comprise the principal information abstractions in the inspection application software design. They can therefore be used to support the design of that software. To illustrate the principles of adopting a model driven approach to design and implementation of an object oriented application, a very simple example is used, the inspection of a "blob" containing "round_holes" as shown in Figure 3.

The Product Design Model shown in Figure 4 describes the object to be inspected as made up of two object types, "blobs" and "round_holes". Both objects have similar attributes apart

from those describing the irregular nature of "blobs" and the circular nature of "round_holes", plus the important relationship that "blobs" have attributes "round_holes". Figure 4 also describes the application software requirements model for Hole Boundaries in an implementation constrained form of a C++ class. This class is an application dependent feature class (as introduced in chapter 5) in this case a specialised type of the boundary descriptor base class. The class contains all the inherited attributes and methods describing the boundary features. What is required at the application level is the addition of a suitable "constructor" which contains the required grey-level and binary matrix level vision processing. The constructor will generate an instance of its class by instantiating grey-level objects and evolving binary and segmented binary objects, and finally extracting its own feature level object, in this case a Hole Boundary object.

FIGURE 3. A Blob containing Holes

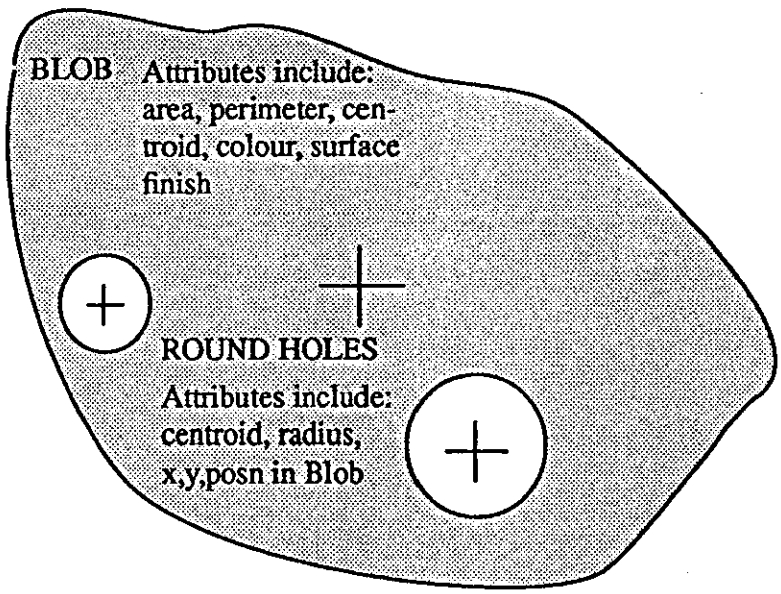
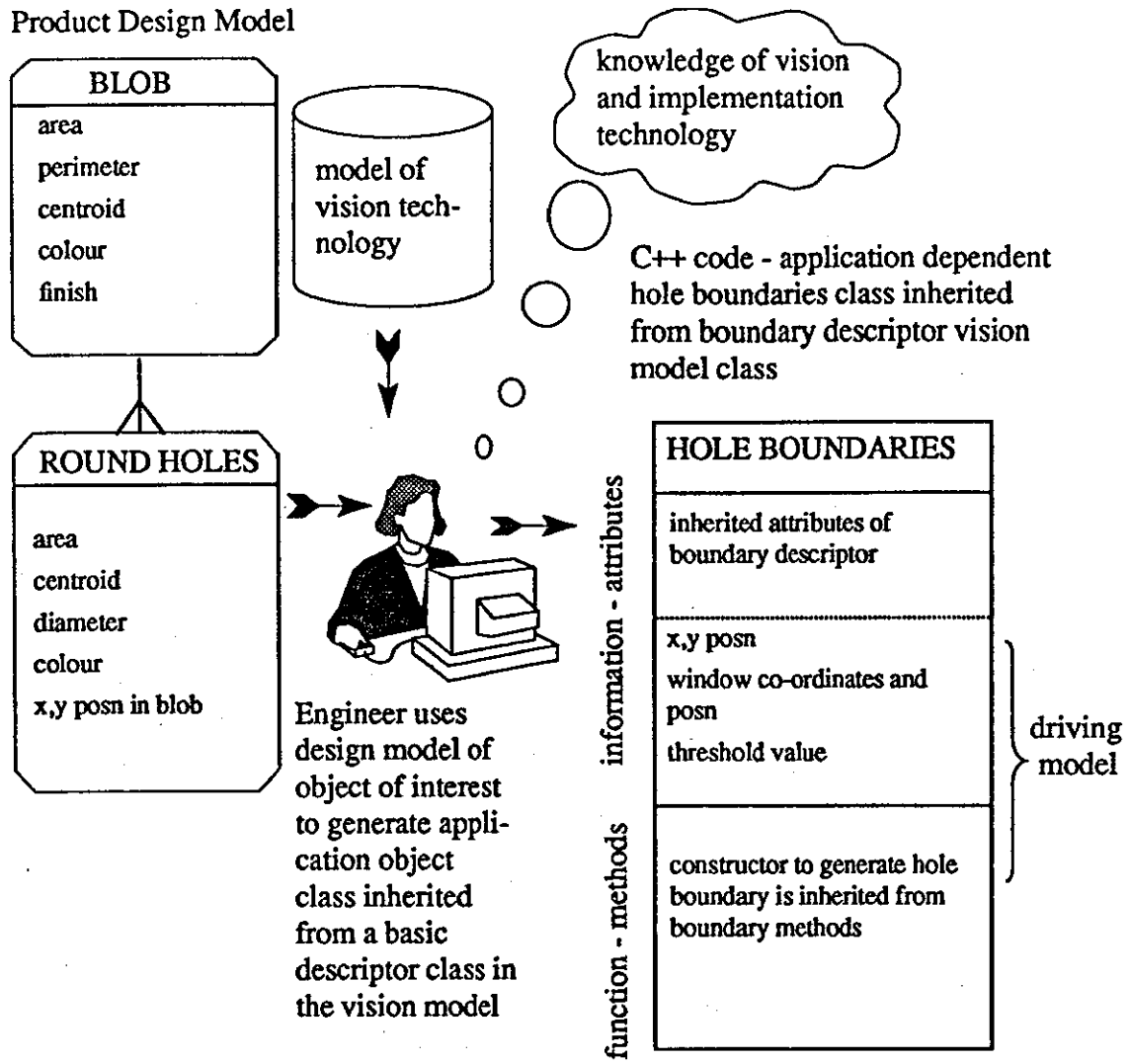


Figure 4 illustrates the idea that view provision, enabling the application software requirements view is done by the design engineer: he/she uses his/her view of the design model of the blob and its attributes, his/her knowledge of the particular vision system implementation, and his/her understanding of image processing supported by the available model of vision technology. Ideally this knowledge and the mechanisms used to encapsulate it will, in the future, be implemented as a case tool.

FIGURE 4. Design model for a blob containing holes used to shape application generation



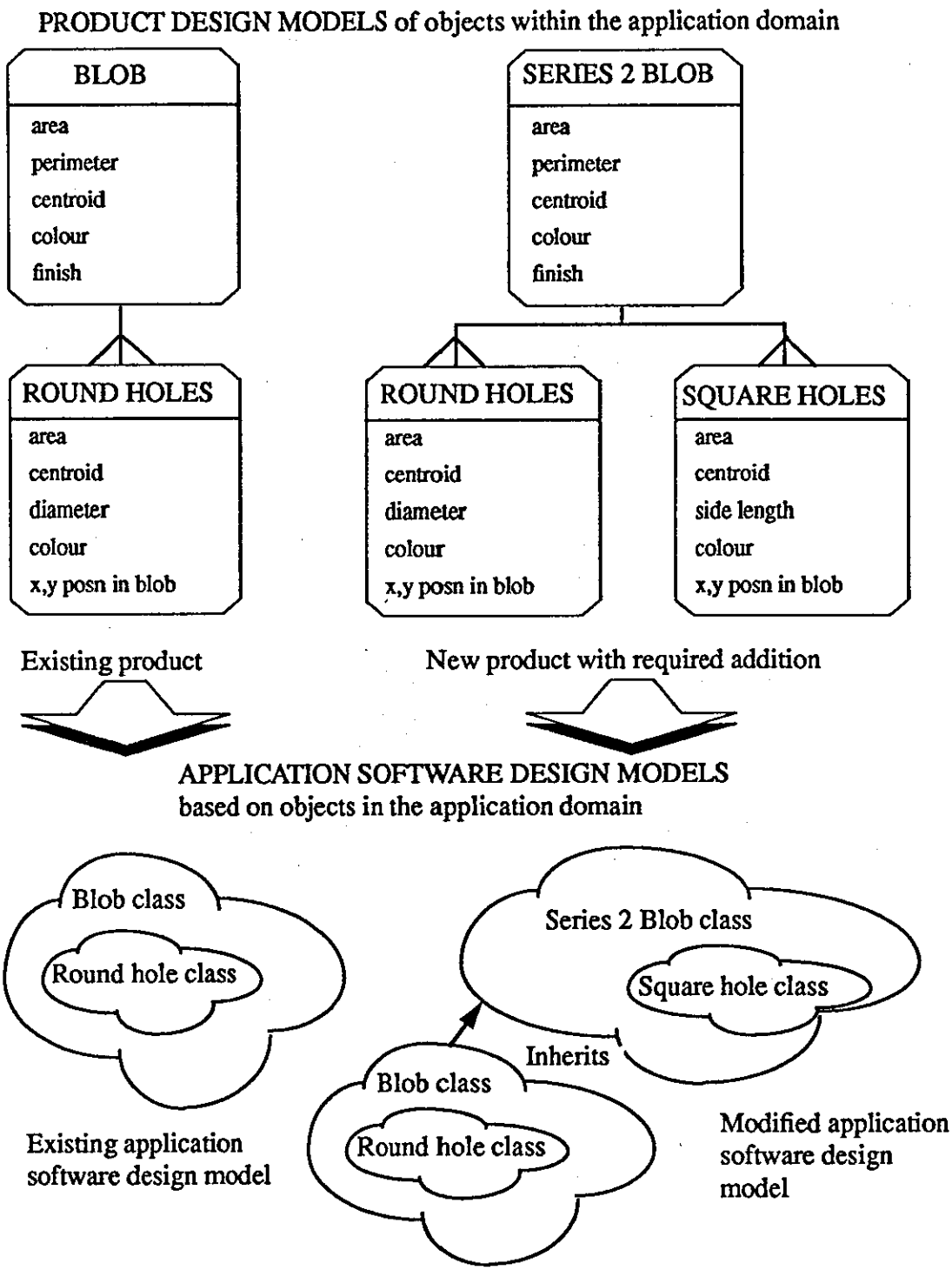
2.4 Modification Of The System

When coupled with the advantages from using object oriented design and object oriented implementation languages [Cox 87, Thomas 89, Meyer 87, Hodgson 90], the tight binding of models of objects within the application domain and the structure of the application software can provide additional benefit. These include the improved potential for software re-use and standardisation, and improved support for software maintenance and change.

Figure 5 demonstrates the use of inheritance to simplify software modification, where there is a requirement to change the original model of a Blob to incorporate a new feature within the model of a new type of Blob. The new model has a second attribute "square holes". To accom-modate this new Blob class, new software is required which can inherit the original Blob class i.e. all the Blob/Round_Hole code, to create a new top level application class. This new class

contains the additional attribute "Square holes" within its private data structure and public methods. The "Square_hole" object class is then designed and built. The application code can be modified with the addition of the new type of Blob class and the new Square_hole class without having to understand or modify the implementation used within the original Blob/ Round Hole code.

FIGURE 5. The use of inheritance to ease software change



The objects within the application domain (i.e. the objects to be inspected) are those objects which have potential for unforeseen change. Therefore, building the structure of the software in line with the structure of the objects in the application domain, enables the potential benefits of inheritance and code modification to be realised. Hodgson [Hodgson 90] makes the important point that "software re-use needs commitment". The structure and control of classes must be carefully designed to enable the potential for re-use. We propose that this can apply equally to realise the potential for "ease of software change", and that the proposed structure based on existing models in the application domain will enhance the potential for ease of change.

An example model decomposition, describing objects in the application domain, is shown in Figure 6. This figure shows a fragment of the "EDIF model for PCB" [EDIF 90, EDIF 91] where the fragment describes a "bare board" entity. The model, and its relevant parts relating to each of the PCB sub entities, identifies an internationally accepted structure. This taxonomy could be used to identify the principal objects and their relationships within application software used to inspect the bare board. The potential for re-use and modification can be enhanced by making reference to, and adopting attributes of, this common structure. If an original application is required only to inspect "physical nets", "fiducial marks" and "mounting places", then these objects would make up the assembly structure of the design. A subsequent requirement might be for the system to inspect "printed components" as their use became increasingly commonplace in the company's products. The required change could be incorporated either by modification of highly structured code which encapsulated previous knowledge of the potential existence of "printed components", or by the inheritance of the "bare board" object class with the addition of the new abstraction "printed components" within the new derived object class.

3.0 A LAYERED ARCHITECTURE FOR STRUCTURING VISION MACHINE APPLICATION SOFTWARE

3.1 Introduction

This section describes a layered architecture to be used within an application. It embraces both the vision model and application specific code. The architecture also recognises the need to provide an interface between the vision hardware resource used in a particular system imple-

mentation and the software which implements an instance of the vision model. The design principles that govern the nature of the architecture are discussed, as are the potential benefits in terms of software re-use and ability to handle hardware heterogeneity.

FIGURE 6. Part of the EDIF model for PCB using IDEF-1X, describing a Bare Board.

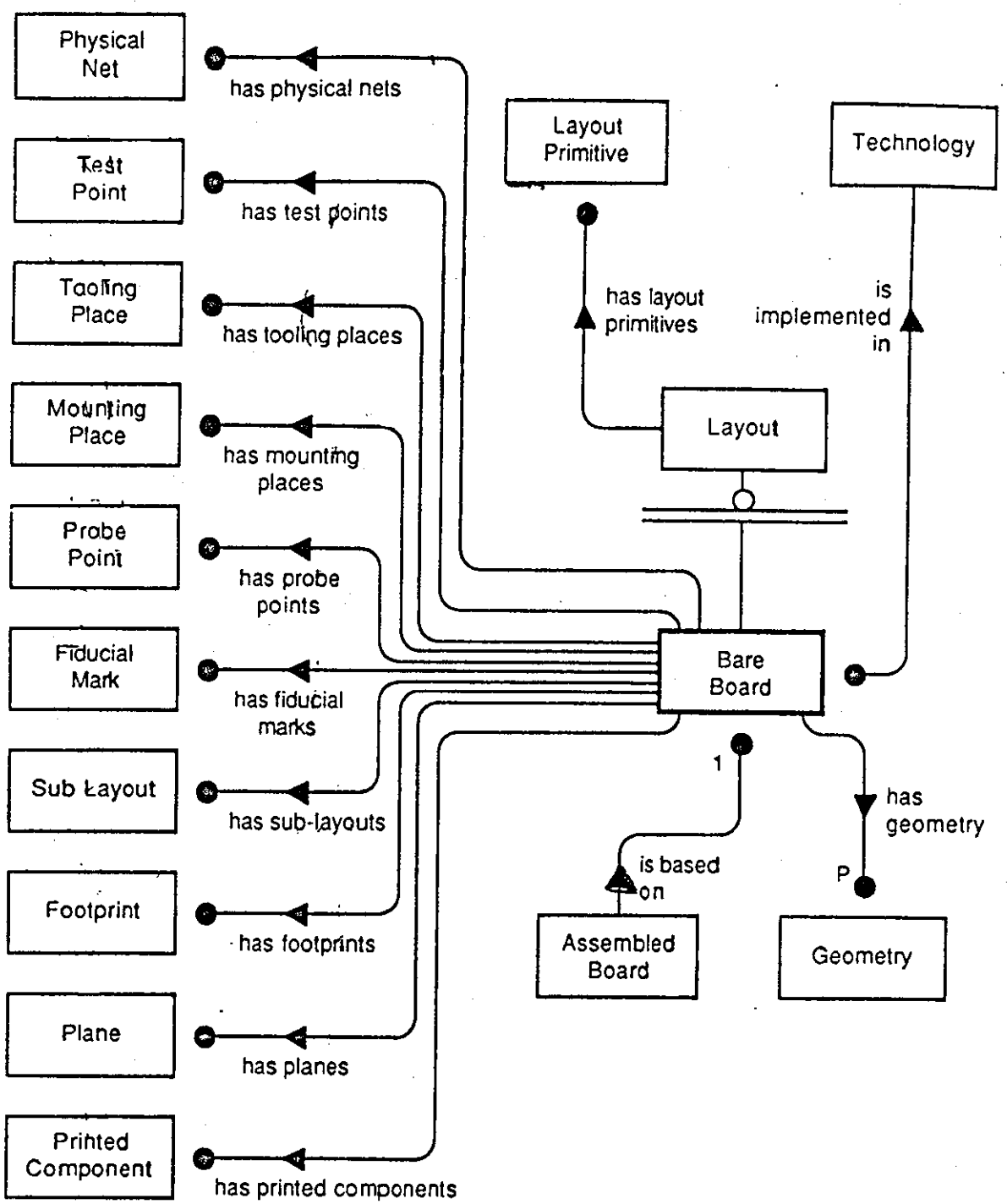


Figure taken from reference [EDIF 91]

3.2 Vision Application Decomposition - Hierarchy Or Heterarchy ?

After many years of research, computer vision is still considered by many experts to be an unreliable technology [OU 92] (i.e. minor changes within the conditions within the viewed scene can cause unpredictable system performance). In all the vision applications built by the author (as described in chapter 4, 7 and 8) the unpredictable performance of computer vision has been a central problem. The capture and processing of consistent matrix images which will generate a predictable set of descriptive features of the object of interest within the application is a primary requirement for the vision system application designer. As computer vision systems do not possess the intelligence which backs up the human eye (despite advances in artificial intelligence [Thomas 91, Mayhew 92]), vision applications must include precautionary processing capabilities to ensure that most common kinds of contingencies can be met. e.g extra objects in the scene caused by spurious noise, uncontrollable changes in lighting conditions, changes in the colour or surface finish of the object to be inspected.

FIGURE 7. A Hierarchical Model of Machine Vision.

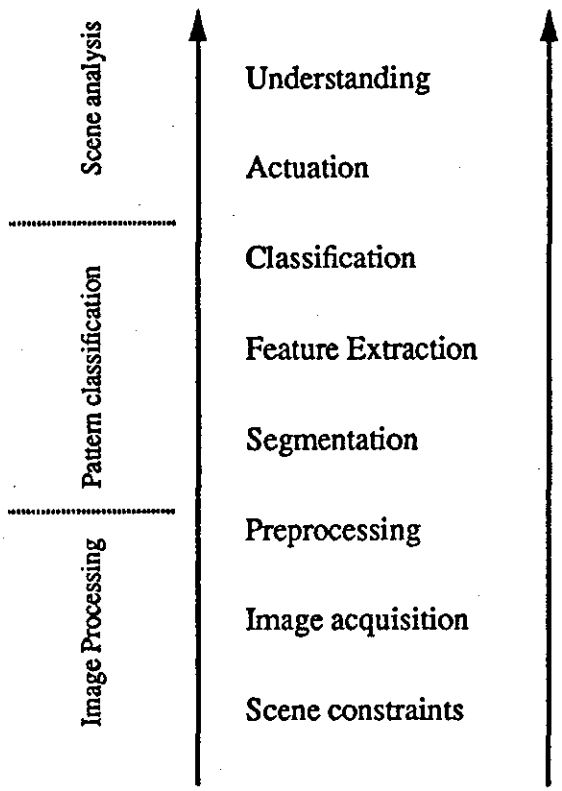


Figure 7 shows a conventional model of image processing [Thomas 91] which describes increased levels of data abstraction, from a complex grey-level image of objects and background to a simple set of features representing an object of interest (which can be classified in

some way and used for scene analysis). The literal implementation of this model as a stack of discrete processes represents an ideal situation. A given application, requiring a description of some object within a scene, would interface with object features and be protected from the details of lower level matrix image processing considerations.

Because of the unpredictable performance of computer vision, implemented applications rarely fit this interpretation of the conventional model (i.e. as a stack of discrete processes), though the categories provide a useful reference, particularly in regard to loosely grouping the classes of image processing, pattern classification and scene analysis.

To ensure robustness, the designer requires the application objects to have complete flexibility in their control over the matrix level objects. Thus application objects are required to view all the data abstractions in a vision model, replacing the notion of a hierarchical processing stack with that of a heterarchy of matrix object classes which can be instantiated and manipulated by the application layer objects. We recognise the heterarchical features described above, and provide realistic mechanisms for system flexibility and support of change.

3.3 A View Of Vision Processing Using A Complex Layered Architecture

Messina and Tricomi [Messina 91] propose a layered stack of discrete processes to mirror those successfully exploited in the ISO OSI model [ISO 78, Mackinnon 90, Pimentel 90]. They argue that a model with standard inter layer interfaces, as detailed in Figure 8, would allow operators and algorithms making up a layer, to be replaced in the vision system in such a way as to leave adjacent levels unchanged. The layers identify groups of homogeneous functions which aim to allow independent processing within layers and minimise the interaction between layers. Typically, the pre-processing level will receive a request for an image with characteristics that can be pre-determined by the next upper level. For example, when the segmentation layer requests pre-processing services, this request triggers multiple operations within the pre-processing layer.

In the author's experience, vision applications require unconstrained access to all functionality within each layer. We believe that a generic model describing specific interlayer relationships of the complexity identified by Messina is unlikely to prove useful in supporting practical machine vision implementations.

FIGURE 8. Messina's proposed model for the robotic vision process

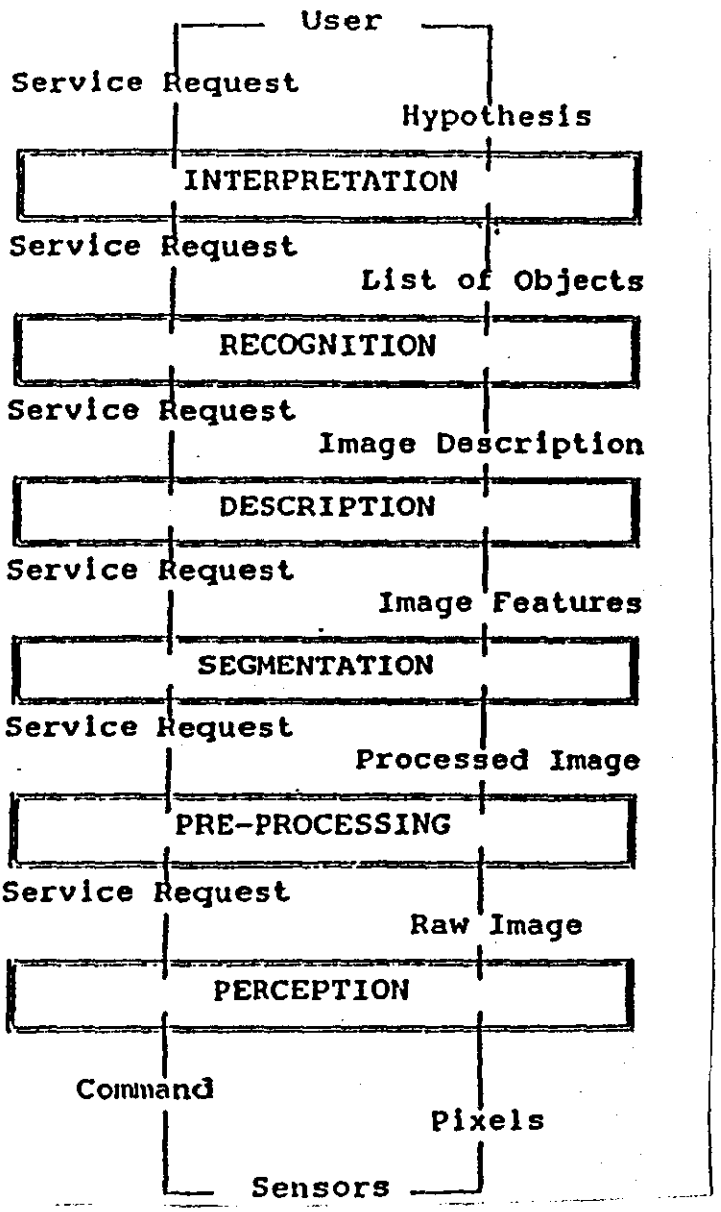


Figure taken from reference [Messina 91]

The author proposes another view of vision processing in the shape of a simple three layered architecture which recognises the needs of applications which are required to extract features, by providing unconstrained but structured access to matrix level vision processing. Matrix level processing and feature extraction are structured within a layer which comprises an implementation of the vision model proposed in chapter 5, while the vision hardware, which often represents the "user to vendor boundary", is encapsulated within a vision hardware resource layer.

FIGURE 9. Two orthogonal views of machine vision processing

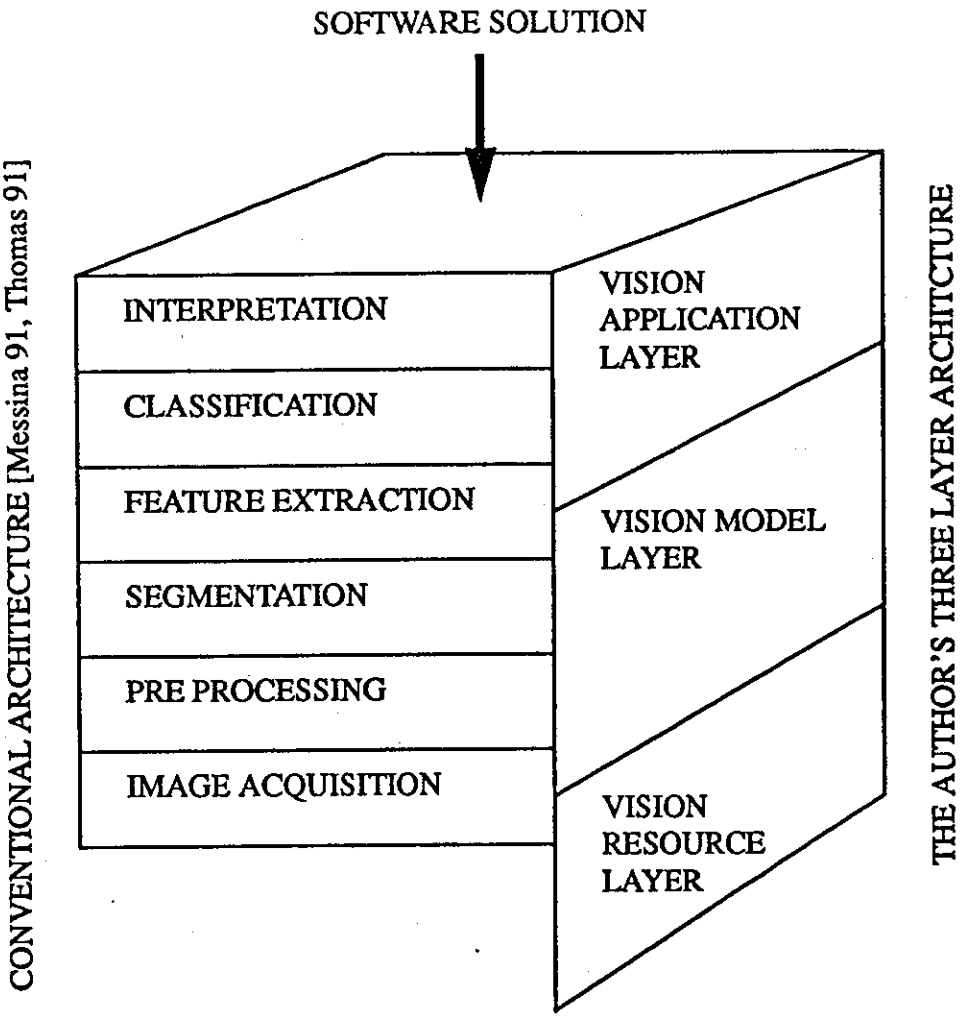


Figure 9 demonstrates how the author's three-layered architectural view can co-exist with the conventional vision architecture. Both architectural views can be of value during vision application analysis and design, while the three-layered architecture can provide additional support during the implementation and maintenance phases of the machine vision system lifecycle.

3.4 The Proposed View Of Machine Vision Using A Three Layered Architecture

The layered architecture proposed here comprises a central model describing vision processing, an application layer comprising the functional requirements of a particular application, and a layer comprising the software required to interface the vision model software to particular vision hardware. The software to hardware interface usually takes the form of a set of software functions contained within a software library supplied by the vendor of the vision

hardware. The vision model lends structure to the vision library functions used in a specific application. The application comprises a structured set of application specific objects, using services offered by the vision model and supplying application specific services to the vision machine user.

Together with Figure 10, the following provides a more detailed description of each layer of the proposed architecture.

3.4.1 The Vision Application Layer

This implements the functionality of the application without concern for how the lower level objects and their methods are implemented. It comprises a hierarchy of interrelated application objects. The objects which border the Vision Model layer utilize the services offered by the vision model layer by instantiating vision objects and controlling them via object methods. The top level object within the application layer offers vision services to the vision machine user.

3.4.2 The Vision Model Layer

This implements a particular instance of the generic model of machine vision, from image capture to feature extraction. It comprises software which conforms to the generic vision model and contains no application specific functionality. A particular implementation of the model is constrained by the language in which it is implemented, and the specification of the vision hardware resource. It offers services to the application layer where application objects instantiate and manipulate vision objects in a manner prescribed by the model. It implements vision processing functionality through an interface to vision primitives. These primitives are in the form of library functions which operate in the hardware resource interface layer, and are dependent on the vision hardware technology.

3.4.3 The Vision Resource Layer

This implements a device-dependent interface between the vision model and a particular type of enabling vision hardware resource. It appears to the model layer as a heterarchy of independent functions, (although it is likely that there will be hierarchical decomposition within the resource layer, as required by the vision hardware vendor). This layer normally

takes the form of a library of function calls controlling vision processing and frame storage hardware and is constrained by the specific nature of this hardware. This layer could also include the use of dedicated vision functions implemented in hardware.

FIGURE 10. Relationships within the layered architecture for Vision application software.

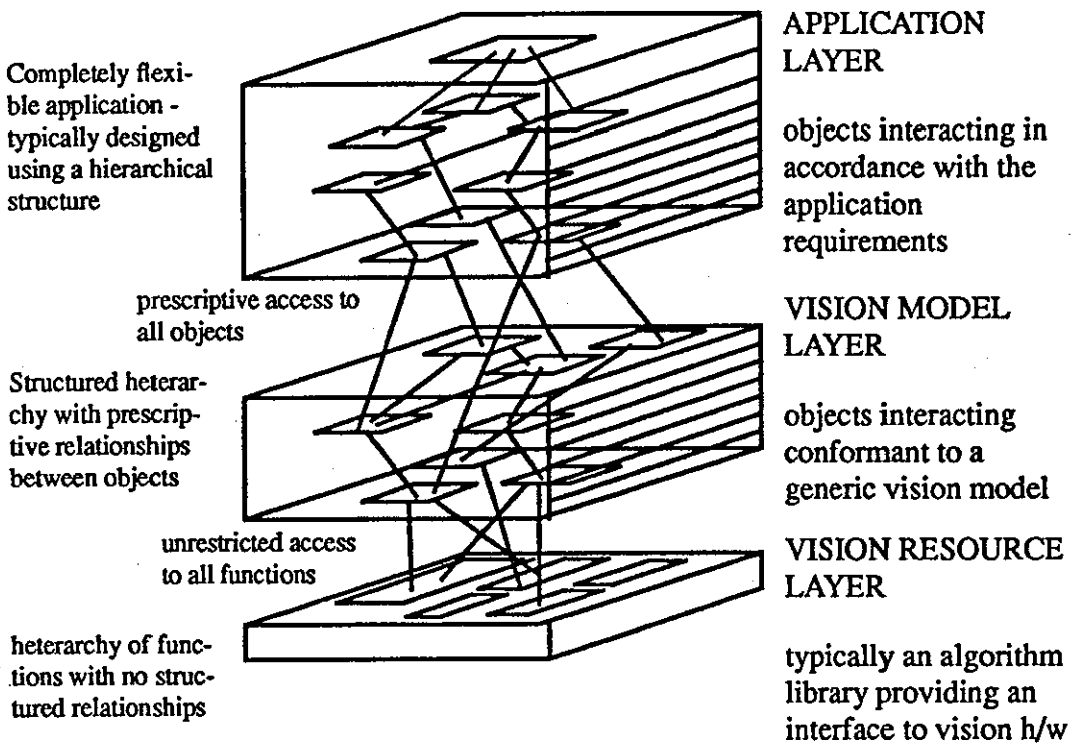


FIGURE 11. A view of the architecture reflecting flexibility

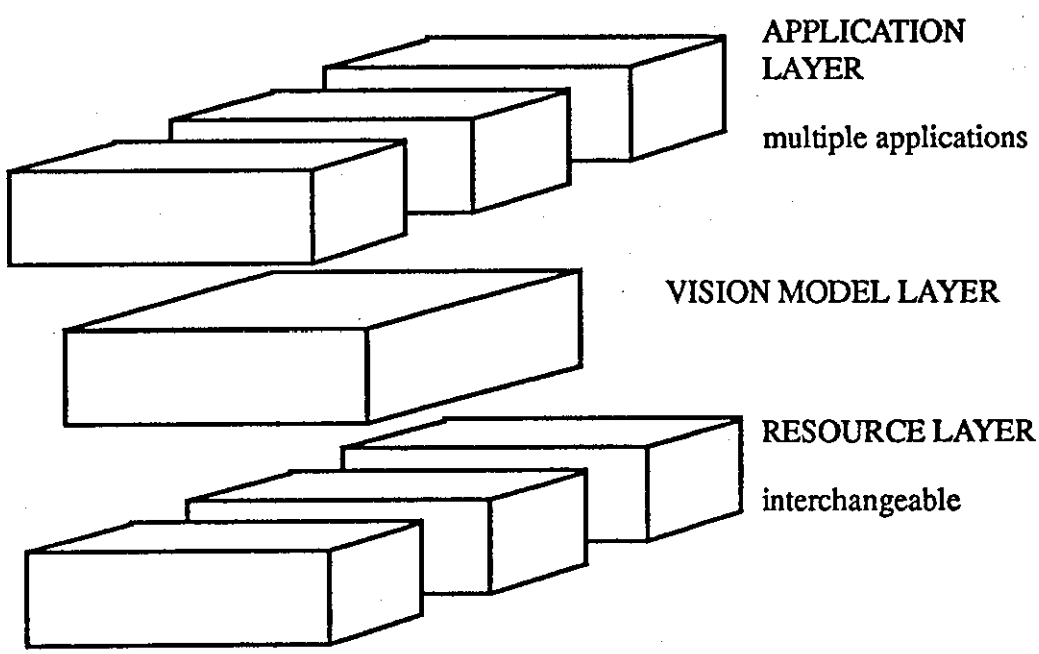


Figure 11 shows relationships between these layers which demonstrate the flexibility offered by the architecture. The vision model can provide services to multiple applications within a single system at a particular point in time, or at different times within the system life cycle. The layer relating to a particular vision resource can be removed and replaced should this be required during the system life cycle. The application analysis and design process can take place without reference to the vision hardware (though detailed design at a low level within the application layer, i.e. application specific feature extraction, is likely to be influenced by the implementation technology). The implementation technology does not impose constraint should the user wish to take advantage of improvements in vision technology. Problems of heterogeneity, where applications are required to run on a range of vision hardware, can also be eased by use of the proposed architecture.

4.0 IMPLEMENTATION MECHANISMS DEFINING THE RELATIONSHIPS BETWEEN LAYERS

4.1 Vision Model Layer To Application Layer

In order for the vision model to be generic, it is important that no application-dependent functionality is present within the model layer. It is this principle, along with the decision that descriptor features are extracted from matrix representations, that forms the vision application layer to vision model layer boundary. Descriptor level feature classes are provided to be used within an inheritance relationship with application-dependent feature descriptors classes. The application-dependent feature descriptors then add the application-specific methods which allow them to perform their required application-specific function.

In order for this relationship to be supported by a C++ implementation, the segmented binary object class must have a friend relationship with the feature descriptor class, or at least with the extraction method of that class. With friend status, the application can now look into the segmented binary image object and extract features without having any access to vision model implementation issues. It is this concept of the application looking into the model, and never the reverse, which allows the model to remain isolated from all application issues. This concept parallels that of Cox [Cox 87] in his description of work on operator interfaces.

4.2 C++ Implementation Examples, Demonstrating The Vision Model To Application Layer Relationship

This section describes the C++ implementation of the relationships discussed above, giving simple examples using application objects from the vision application domain of "component inspection".

FIGURE 12. code fragment showing .hpp file for the class SIMPLE_ARC

```
#include "visl192.hpp"
#define UTINY unsigned char
//*****************************************************************************
//       name: simple_arc_descriptor
//         is a : base object class
//  objective : contains chaincode info to desribe arc between two points
//   friend to : binary_image
// written by : John Edwards 90*
//       mods ; JE 92
//*****************************************************************************
class simple_arc_descriptors {
protected:
    int   proc_stopgo;  // STOP OR GO
    UTINY *chainbuf; // ptr to buffer holding chaincode values
    int   no_ellems;    // initial no of elements for chain
    int   xstart;       // x coordinate of arc start posn
    int   ystart;       // y coordinate of arc start posn
    int   chaincount;  // number of points in the arc point set
    int   xfinish;      // x coordinate of arc finish posn
    int   yfinish;      // y coordinate of arc finish posn
    long  arc_length;  // length of arc
*/
the following attributes are passed from the segmented binary image
from which the arc is extracted. They are window atributes which are not strictly attributes
of an arc but are required in the arc extraction process. The initialise function is made a friend of
the segmented binary image class in order to access its private data and set up these values.
This forms the link between the vision model and application specific feature object classes.
from which the arc is to be generated. They are copied and not just accessed by each method using
pointers because it is nessessary to derive subclasses from arc which allso needthis info eg boundary,
otherwise they would all have to become friends of binary image which is best avoided.
/*
      int    imagebuf;  // FB0 OR FB1
      int    workbuf;   // FB2 OR FB3
      int    xorigin;   //top left hand corner x coordinate
      int    yorigin;   //top left hand corner y coordinate
      int    width;     //width distance in x - 512
      int    height;    //height distance in y - 480
public:
      void initialise_from_segmented_binary_image_friend(segmented_binary_images *a_segmented_binary_image);
      virtual int find_start();
      int get_arc();
      int check_start();
      int extract_a_simple_arc_from(segmented_binary_images *a_segmented_binary_image);
      int get_length_of_your_arc();
      int vdu_display(char *arc_name);
      int save_to_disc(char *file_name,int ident_no);
      int get_from_disc(char *file_name);
      int arc_to_monitor(int newx,int newy);
//    ~simple_arc_descriptors() { delete[chaincount] chainbuf; }
};
```

Consideration is first given to the requirements of the base descriptor objects within the vision model which are to be used, within inheritance relationships, to form application layer objects. As stated above, these classes are required to extract features from segmented binary objects. To achieve this they require access to information which will be private to the segmented binary object. It is for this reason that the extraction method of a descriptor class must be made a friend of the segmented binary class.

FIGURE 13. code fragment showing part of the .cpp file of method EXTRACT_A_SIMPLE_ARC

```
#include "sarc92.hpp"
//*************************************************************************
//    Method name : extract_a_simple_arc()
//    Objective : save image , find a start, extract the arc
//                check if the start posn was good, if not use
//                end point as a new start point and try again
//    parameters : a_simple_arc_descriptor - name of the arc
//*************************************************************************
int simple_arc_descriptors::extract_a_simple_arc_from(segmented_binary_images *a_segmented_binary_image)
{
    int checkok;
    int loopout=0;

    a_segmented_binary_image->protect_image();

    this->initialise_from_segmented_binary_image_friend(a_segmented_binary_image);

    if (this->find_start()==FALSE) return(FALSE);
    do
        {
        if (this->get_arc()==FALSE) return(FALSE);
            checkok = (this->check_start());
            if (loopout++ >5)
            {
                printf("can't find good start");
                return(FALSE);
            }
            if (checkok == FALSE) a_segmented_binary_image->restore_image();
        }
    while (checkok == FALSE);
    return(TRUE);
}
```

Figure 12, shows the header declaring the protected data and the public methods of the "SIMPLE_ARC" descriptor class. Comments in the code explain the necessity to gain access to the private information of a segmented binary object. The information required is that specifying where the matrix level information resides, i.e. in which image buffer and what window parameters the segmented binary image exists. Accessing this information is done through the method "INITIALISE_FROM_SEGMENTED_BINARY_IMAGE_FRIEND". The use of this method can be seen by examining the code in Figure 13 which shows the implementation code required to extract a simple arc. This method "EXTRACT_A_SIMPLE_ARC" is an aggregation of methods within its own "SIMPLE_ARC" class and uses the "this->" operation

to invoke methods of its own class. It can be seen from the figure that it initialises the "SIMPLE_ARC" object, thus gaining access to the whereabouts of the matrix information. It then finds the startpoint of the arc and extracts it.

Thus implementation entirely reflects the required principle of extraction of descriptor features from matrix objects while providing the highest degree of separation between the two classes of image processing objects.

Figure 14 shows the header code describing an application level object class. The class declaration (immediately after the commented heading) shows that this class is inherited from the descriptor level class "BOUNDARY_DESCRIPTORS". In this simple example the class has only two methods - a constructor (having the same name as the class name) and an application dependent method (which outputs the class data to the screen). All methods of the inherited base class "BOUNDARY_DESCRIPTORS" are available to the inheriting application level class. The use of these inherited methods can be seen by examining the code listed in Figure 15.

FIGURE 14. Code fragment showing the header for An application layer class UNCLASIFIED_COMPONENT_BOUNDARY

```
#include "bound92.hpp"// includes sarc92.hpp -> visl192.hpp

//*************************************************************************/
//    APDES92.CPP    OO PROGRAM HEADER  IN ZORTECH C++
//
//    CONTAINS HEADER CODE FOR THE FOLLOWING
//    APPLICATION LEVEL DERIVED FEATURE CLASSES
//
//    COMPONENT_BOUNDARY derived from BOUNDARY
//    note the constructures contain the grey and
//    binary level vision processing required prior to
//    the extraction of arcs or boundarys
//*
//    WRITTEN BY JOHN EDWARDS DATE April  92 /
//*************************************************************************
class unclassified_component_boundarys : public boundary_descriptors {

private:
//      attributes inherited from boundary_descriptors class
//      typically area, perimeter, centroid, chaincode see BOUND92.HPP

        int ident_no;        // object identification number
public:
//      class methods
//      contructor
        unclassified_component_boundarys(grey_images *a_grey_image,int ident,int threshold);
        void display_your_feature_data();
};
```

FIGURE 15. code fragment showing part of the .cpp file for the constructor method "UNCLASSIFIED_COMPONENT_BOUNDARIES"

```
#include "apdes92.hpp"

//*************************************************************************/
//   APDES.CPP   OO PROGRAM IN ZORTECH C++
//   CONTAINS   IMPLEMENTATION CODE FOR THE FOLLOWING
//   APPLICATION LEVEL DERIVED FEATURE CLASSES
//
//   COMPONENT_BOUNDARY derived from BOUNDARY
//   note the constructures contain the grey and
//   binary level vision processing required prior to
//   the extraction of arcs or boundarys
//
//   WRITTEN BY JOHN EDWARDS  DATE APRIL 92
//*************************************************************************
unclassified_component_boundarys :: unclassified_component_boundarys(grey_images *a_grey_image,int ident,int threshold)
{
 int error_return = 1;
 ident_no = ident;

   if (a_grey_image->spatial_filter1_your_image() == FALSE) error_return = -1;

   binary_images    a_binary_image;
   if (a_grey_image->threshold_your_image_to_evolve_a_binary_image_called
         (&a_binary_image,0,0,threshold,255,255)==FALSE) error_return = -1;

   if (a_binary_image.erode_your_image(2,BLACK)==FALSE) error_return = -1;
   if (a_binary_image.dilate_your_image(2,BLACK)==FALSE) error_return = -1;

   segmented_binary_images a_seg_bin_image;
   if (a_binary_image.use_a_boundary_to_evolve_a_segmented_binary_object_called
         (&a_seg_bin_image,WHITE)==FALSE) error_return = -1;

   if (a_seg_bin_image.clear_your_border_to_this_colour_and_size(BLACK,5)==FALSE) error_return = -1;
   if (a_seg_bin_image.frame_your_border_with_this_colour_and_size(BLACK,5)==FALSE) error_return = -1;
   if (a_seg_bin_image.protect_image()==FALSE) error_return = -1;

   if (this->extract_a_boundary_from(&a_seg_bin_image)==FALSE) error_return = -1;

   if (a_seg_bin_image.restore_image()==FALSE) error_return = -1;

   if (this->directly_get_your_centroid_and_display()==FALSE) error_return = -1;
   if (this->get_your_perimeter()==FALSE) error_return = -1;
return(error_return);
}
```

In the code fragment shown in this figure, the application object, having been passed a grey_-image object, applies particular grey-level image processes to it. It tells the grey_image object to evolve a binary_image object. It then applies particular image processes to the binary_image, which it then requests to evolve a segmented_binary_image. The constructor can now extract its required boundary using the methods inherited from the boundary descriptor object class through the "this->" operation. The extraction is followed by the use of more inherited methods, to compute centroid, area and perimeter features from the extracted boundary.

The simple applications described above show how the application builder has control over all abstractions in the vision processing domain but is protected from the complexities of the vision processing implementation. The application builder is provided with a set of base classes, or a class library, which reflect the information, function and object relationship abstractions of the vision model.

4.3 Vision Model Layer To Vision Resource layer.

Instantiation of the vision model layer is constrained in two ways:

- by its programming language implementation, and
- by the form of vision processing hardware, as this hardware defines the form of the software which interfaces the vision model to the vision hardware.

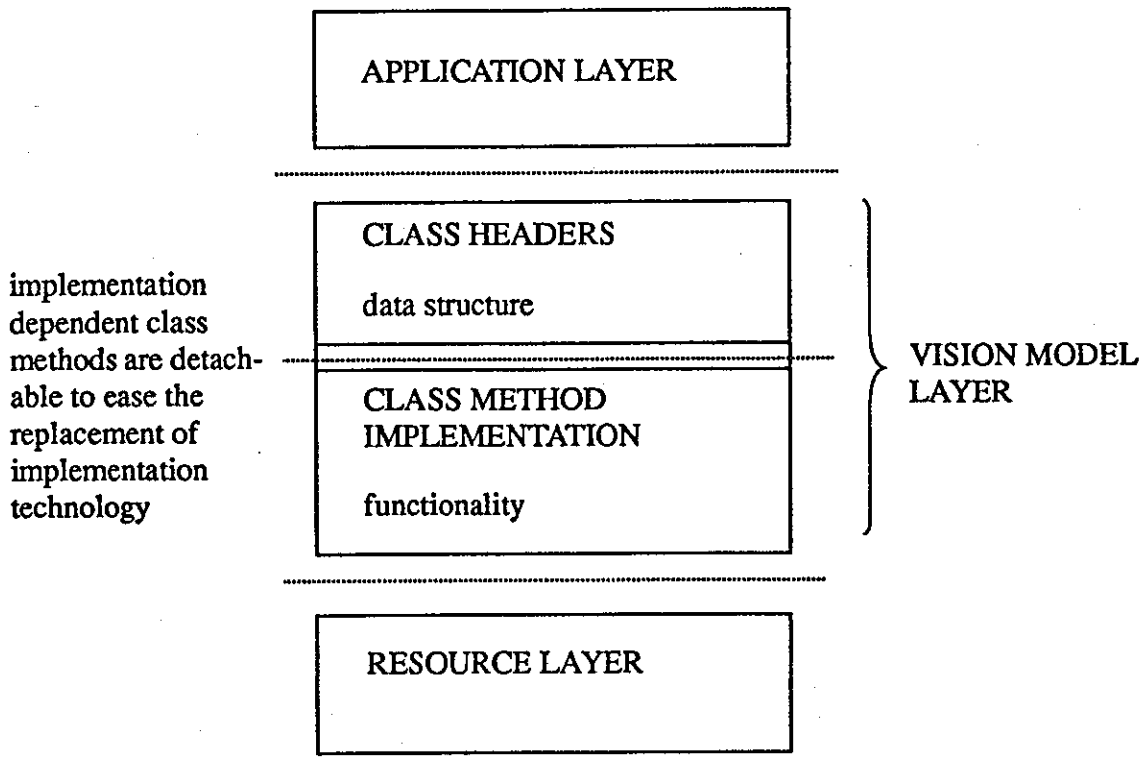
The vision resource used within this study comprises image storage hardware in a host processor, with dedicated software (in the form of a library of functions) executed on the host processor.

The boundary between the vision model layer and the vision resource layer exists in the methods of the vision model object classes, as they call the vision resource layer functions. The methods therefore depend on the vision resource technology.

The object oriented implementation of the vision model, using C++ object classes, provides a practical, "clean" interface between the definition of a class and its functional implementation. A C++ Class is defined by declaring its data abstraction and set of methods in a Class header file, while the method implementation, comprising the C++ code exists in a separate but related file. This separation supports hardware resource heterogeneity through re-implementing the vision model methods for particular vision hardware resources as required. Where appropriate (in similar types of vision implementation technology) most of the code would be re-used. A functional specification of each method must be identified, and compliant methods must be developed for new vision hardware resources. The application layer only interfaces to the vision model class library headers, which remain unchanged for all implementations. Therefore the application layer can remain unaffected by any change in vision resource technology. Figure 16 shows a representation of this decomposition, where the object methods

within the model form convenient detachable code elements which are replaced when replacing the vision hardware resource.

FIGURE 16. **Vision model layer to implementation layer interface**



Chapter 7

THE DESIGN AND BUILDING OF AN INFORMATION DRIVEN APPLICATION OBJECT FOR AUTOMATED VISUAL INSPECTION

1.0 INTRODUCTION

This chapter demonstrates the application of the models, methodologies and mechanisms proposed in this thesis for the design and implementation of a discrete vision application object. The object will comprise one part of a distributed vision inspection solution within a Soft CIM Cell. The techniques demonstrated in this chapter can be summarised as follows:

- the use of information models to drive applications at run time;
- the use of information models to guide application design;
- the use of a three- layered architecture within the vision application object system implementation;
- the use of a generic model of matrix level vision processing and feature extraction;
- the use of the object oriented paradigm within design and implementation.

2.0 THE REQUIREMENTS FOR THE APPLICATION OBJECT

2.1 Top Level Decomposition Of A Vision Application

The top level objects describing the overall requirements of the application are shown in Figure 1. The application is designed to populate a product model. It uses design information to support the generation of live information from surface mount components. The design information and generated live information is stored in a formal model format. The model

used is the industry standard EDIF model for PCB [EDIF 90,91]. Details of the model and its use within the application are sketched in this chapter.

The objects identified in Figure 1 comprise:

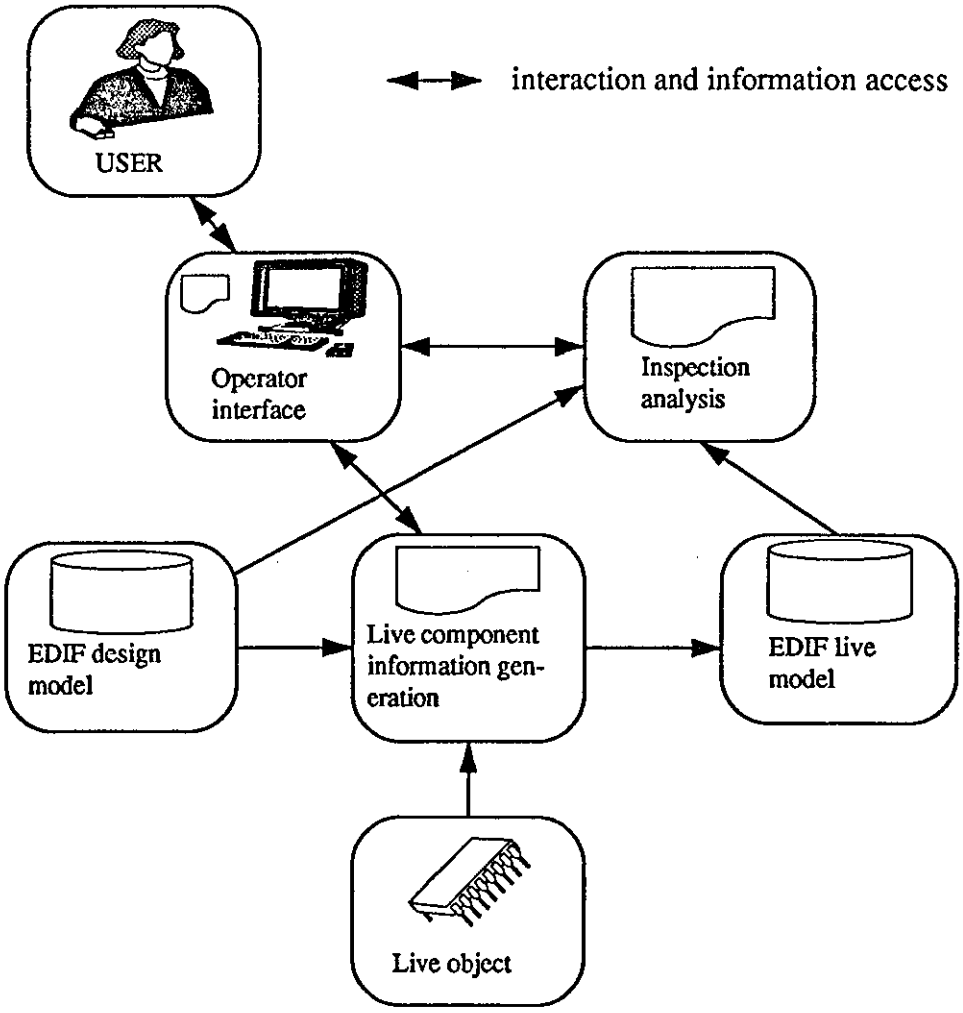
- the "design model" and "live model" objects, which exist as information objects within the user's enterprise information system. Information services are available from the integrating infrastructure;
- the "operator interface" object, which will be implemented on a workstation terminal together with other computer processes required within the production cell. Interaction with the operator interface will be via interaction services available from the integrating infrastructure;
- the "live component" object, which will comprise hardware and software to provide a digital representation of the live component. The interface to this will be particular to the equipment manufacturer. The proprietary vision hardware and software handling is provided through specific implementation of the vision model object methods and vision hardware resource layer, within the 3 layered architecture proposed in chapter 6;
- the "live component model information generation" object comprises the main component of the model driven application described within this chapter;
- the "inspection analysis" object has a subset of the interaction requirements of the live component model information generation object. It is not discussed further in this thesis.

This chapter details the design and implementation of the "live component information generation" object which has the following primary requirements:

- to capture an image of a component, extract a description of the component and convert it to the EDIF compliant model form;
- to take advantage of the provision of a library of component design descriptions, available in EDIF model form.

In order to take advantage of design information whose availability is supported by services from the integrating infrastructure the application object is based on information driven principles [Schildt 87,Weston 92].

FIGURE 1. Top level object modules within the application domain



The technique used in the vision application described in this chapter adds greater flexibility by being able to cope with inspection in unpredictable circumstances. It takes advantage of the fact that a restricted set of attributes can be used to describe the objects which are to be inspected. This enables the classification of components which arrive for inspection in an unpredictable order. The system requires only the location of prospective data stores in which to look for information, and takes advantage of the ease of access to the host information system by services offered through the integrating infrastructure.

3.0 THE EDIF MODEL FOR PCB

3.1 Introduction

The proposed industry standard model for describing PCB's is the "EDIF model for PCB" [EDIF 90,91]. EDIF was originally created for the interchange of electronic design data in

VLSI design systems. Since that time, its use has been extended to enable interchange between a greater range of different systems (e.g. CAE/CAD, manufacture, test), where information transfer requires some electronic mechanism, typically, a disc or a communications network. The EDIF PCB Technical Sub-Committee is addressing aspects of the EDIF standard associated with the design, manufacture, assembly and test of PCB's. The EDIF technical Sub-Committee has a brief to create a conceptual model of a PCB which conveys the information required within the life-cycle of a PCB as defined below:

schematic entry CAE - specification of PCB functionality in schematic form;
component selection - component selection based on technology;
simulation - computer based verification of the specification;
pre-allocation - preliminary allocation of functions into components;
layout - component placement, routing, gate and pin swapping
layout analysis - computer based simulation and verification of the layout;
post processing - plotting, artwork generation;
manufacturing - production of the bare board;
assembly - mounting of components onto the bare board;
test - several types of test applied to the bare board and to the finished board.

The version of the model used in this thesis is EDIF Version 2 2 0, which is based on modelling work that concentrates on design layout. Machine vision applies during PCB manufacture, assembly and test. Critical information required during these processes may well be absent from the version 2 2 0 model. Recommendations for modification to the model to embrace the needs of automated inspection of components are proposed in this chapter, and used in the demonstration application.

The conceptual model is described using a slightly modified version of the IDEF-1X diagramming tool.(A description of the modifications is included in reference EDIF 90). A textual description of an information model of EDIF Version 2 2 0 [EDIF 91] using the EXPRESS information modelling language was used in the thesis.

Within the model, the designer of the vision inspection application is looking for two things:

- the information appropriate to drive the application at run-time;
- principal objects and domain (EDIF) generic structure within the model that can be used to provide non-rigorous structure to the application software design.

3.2 EDIF Model Driven Component Inspection

Within the EDIF model, electronics components such as small outline integrated circuits (SOIC's), small outline transistors (SOT'S), or chip capacitors which will be mounted on PCB's during the assembly operation, are described within the Part Library Section. Information in a particular Part Library Section would be populated from electronic component manufacturer's data sheets. The entity relationship diagram describing packages shown in Figure 2 is taken from the EDIF model IDEF-1X documentation [EDIF 90]. The entities shown with dotted outlines belong to a different model section and are shown in the figure to indicate their interaction with entities in the Parts Library section. The vision application is concerned with the inspection of the physical packaging of the component or part. It is therefore necessary to navigate to the "Package" entity which is described within the Package Library Section. A more detailed representation of the Package is shown in the EXPRESS information model in Figure 3. In this figure, the taxonomy covers a complete decomposition of the Package entity, from Package down to the primitive geometric representation of a set of related points describing the physical shape of the component package.

FIGURE 2. The EDIF entity relationship diagram describing PACKAGES

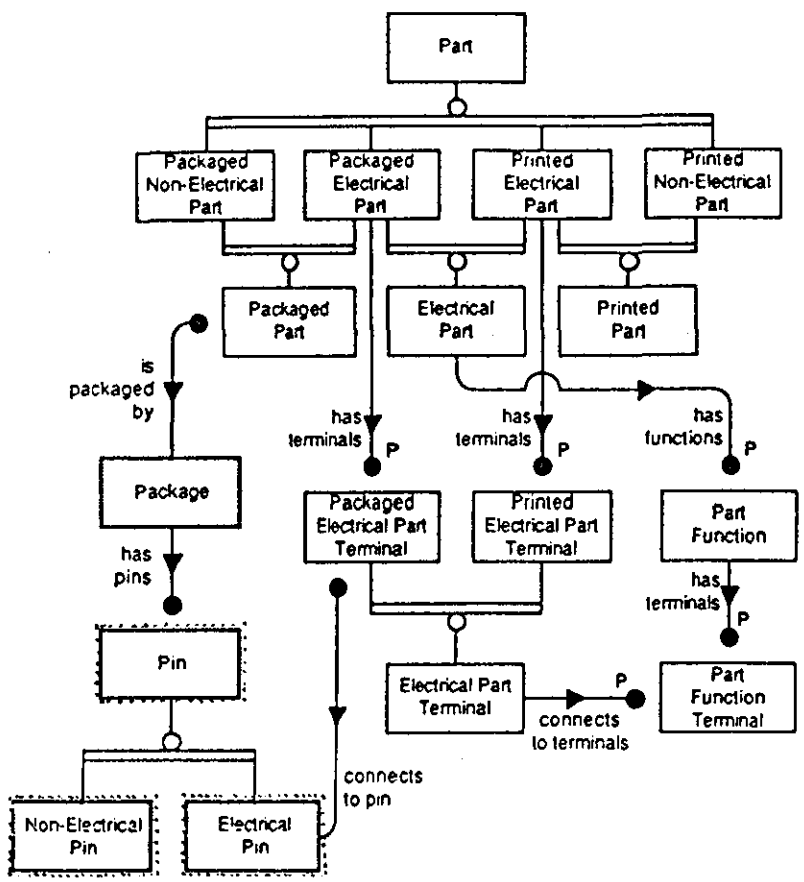


FIGURE 3. The EXPRESS information model of the EDIF representation of PACKAGES

```
ENTITY package;
      has_pins : SET [0 : #] OF pin;
      id : INTERNAL ID_STAMP;
      acronym : OPTIONAL INTERNAL JEDEC_ACRONYM;
      package_class : INTERNAL PACKAGE_CLASS;
      body_volume : INTERNAL THREE_D_SPEC;
      occupied_space : INTERNAL THREE_D_SPEC;
      true_shape : OPTIONAL INTERNAL SET [1 : #] OF THREE_D_SPEC;
WHERE
      only_occurs_in_relation_to :
         valid_users (PACKAGE,
                      ['PACKAGED_PART.IS_PACKAGED_BY',
                      'MOUNTABLE_PACKAGE.USES_PACKAGE']);

      can_exist_alone :
         can_exist_alone (PACKAGE, TRUE);
END_ENTITY;


ENTITY pin
      SUPERTYPE OF
         (ONEOF (non_electrical_pin, electrical_pin) AND
         ONEOF (circular_pin, rectangular_pin));

      name : INTERNAL IDENTIFIER;
      point_position : INTERNAL PACKAGE_POINT;
      pin_length : INTERNAL REAL;
      pin_group : INTERNAL PIN_GROUP_NAME;
DERIVE
      referencing_package :
         package := back_ref (PIN, 'PACKAGE.HAS_PINS');
WHERE
      only_occurs_in_relation_to :
         valid_users (PIN,
            ['PACKAGE.HAS_PINS',
            'MOUNTABLE_NON_ELECTRICAL_PIN.USES_NON_ELECTRICAL_PIN',
            'PACKAGED_ELECTRICAL_PART_TERMINAL.CONNECTS_TO_PIN',
            'MOUNTABLE_ELECTRICAL_PIN.USES_ELECTRICAL_PIN']);

      can_not_exist_alone :
         can_exist_alone (PIN, FALSE);

      depends_upon_one :
         used_by_one (PIN, 'PACKAGE.HAS_PINS');
END_ENTITY;


ENTITY THREE_D_SPEC;
      figure : INTERNAL POLYGON;
      height : INTERNAL Z_DISTANCE;
WHERE
      the_height_is_positive :
         height > 0;
END_ENTITY;

ENTITY POLYGON;
      line_segments : INTERNAL LIST [3 : #] OF LINE_SEGMENT;
WHERE
      the_line_segments_do_not_cross : BOOLEAN := to_be_written;
END_ENTITY;


ENTITY LINE_SEGMENT;
      start_point : INTERNAL POINT;
      end_point : INTERNAL POINT;
END_ENTITY;


ENTITY POINT;
      x : INTERNAL X_DISTANCE;
      y : INTERNAL Y_DISTANCE;
END_ENTITY;
```

Enhancements to the EDIF model are proposed to enable the model to support automated vision inspection of package pins. This requires the Pin entity to have attributes describing the physical shape of the pin. The information contained in the new model representation can be used to aid the extraction of live package pin features. Figure 4 shows details of the proposed new Pin entity. The attribute "Pin Length" shown in Figure 2 has been replaced by the attributes "occupied space" and "true shape" which enable the pin to be modelled in three dimensions.

To populate such a model with live information, the application has to extract appropriate features from the live image of a component. These features can be used to determine the primitive points describing the physical shape of the package. To ease the processing required to achieve this feature extraction, and to increase its reliability, knowledge of the structure of the package held within the Package Library section of the EDIF model is used to drive the application. The design position of each geometric point of interest, relative to the design centroid of the package is computed. This point is used as the centre of a processing window, generated relative to the centroid of the live package. The window is used to enable a restricted search for the live feature.

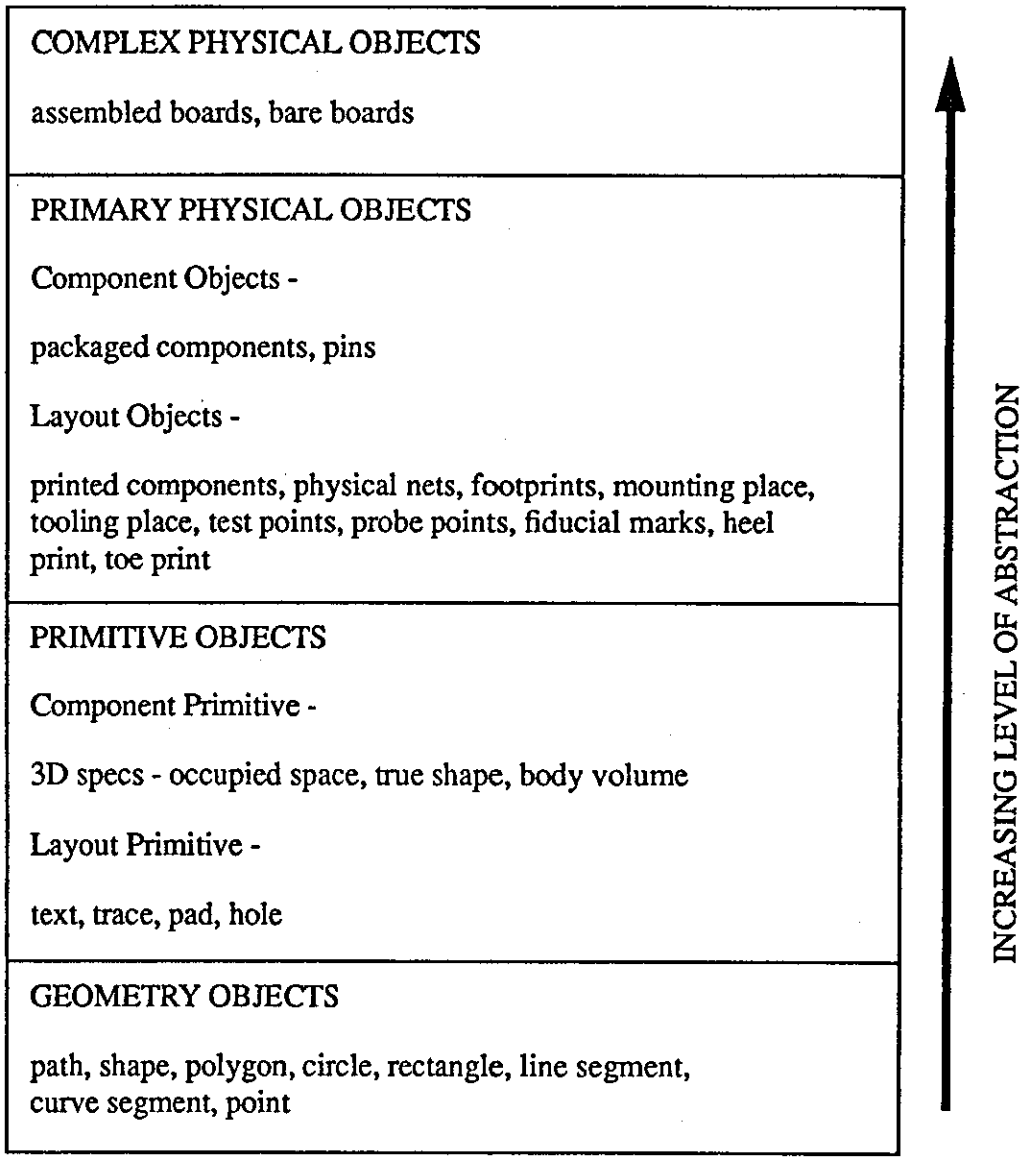
FIGURE 4. Proposed new EDIF PIN Entity to support automated inspection of package pins

```
ENTITY pin

SUPERTYPE OF
    (ONEOF (non_electrical pin, electrical pin) AND ONEOF (circular pin, rectangular pin));
    name : INTERNAL IDENTIFIER;
    pin_group : INTERNAL PIN_GROUP_NAME;
    point_position : INTERNAL PACKAGE_POINT;
    occupied space : INTERNAL THREE_D_SPEC;
    true_shape : OPTIONAL INTERNAL SET[1:#] OF THREE_D_SPEC;

DERIVE

as original v2 2 0

WHERE

as original V2 2 0

END_ENTITY
```

3.3 The EDIF Model Within The Application Design

Automated visual techniques are appropriate for the inspection of the physical attributes of a PCB. Therefore the sections of the conceptual model for PCB that are of interest are those that describe physical entities.

FIGURE 5. A common layered decomposition within the physical entities of the EDIF model



Examination of the sections of the conceptual model for PCB, which describe physical entities yields a taxonomy similar to that describing Packaged Parts as detailed in Figure 3. Appendix 3 gives further examples of this taxonomy within objects which would be useful in the inspection of bare PCB's. Appendix 3 details Layout entities, typically Test-Point and Mounting-Place, which comprise a set of Layout Primitive entities such as Pad and Hole. Primitive Lay-

out entities further decompose into geometry entities. The common taxonomy within the physical entities of the EDIF model for PCB can be described using a layered architecture as detailed in Figure 5. Within the two layers describing primary and primitive physical objects, the layers can be further decomposed into component objects used to describe electronic components, and layout objects used to describe the bare PCB. Layout and component primitive objects are described using a set of "Geometry" entities.

This architecture is used within the demonstration system to structure the principal objects and their relationships when considering an information view of the problem domain.

4.0 THE DESIGN OF THE MODEL DRIVEN APPLICATION OBJECT

4.1 The Operation Of The Vision Machine

The scope of the application described in Figure 1 shows the principal abstractions and their interaction. A description of the operation of the system follows.

The operator will request inspection to begin. The system will then:

- extract a description of the unknown component in terms of a set of elementary features. Within the restricted domain of the application, the area and perimeter features will enable classification of the component type. The centroid is used as position information to enable detailed feature extraction to take place on a component placed anywhere in the field of view of the vision sensor;
- search the available information system for a component which matches this elementary feature description (the component descriptions are held in EDIF format);
- extract the knowledge (from the information model of the matched component) required to drive the process of extraction of detailed information from the component pins, and;
- use this knowledge to extract a detailed description, or live model, from the component under inspection, and store the live description of the component in EDIF form as a contribution to the dynamic aspect of a product model.

For the first version of the application, live information is required for the Component Package Pins only.

FIGURE 6. The "live" component generation object module and its surrounding environment

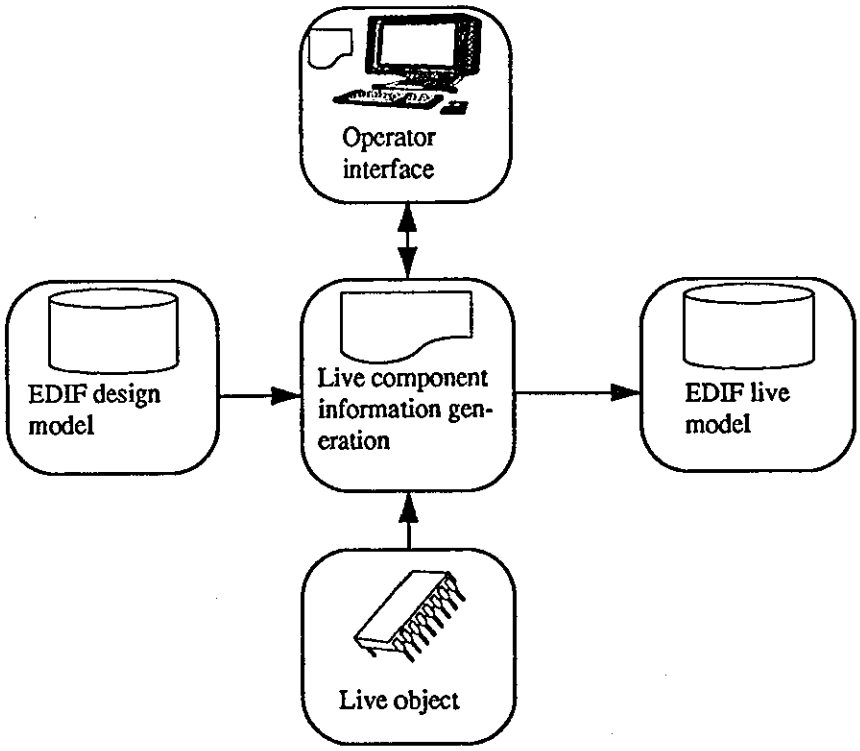
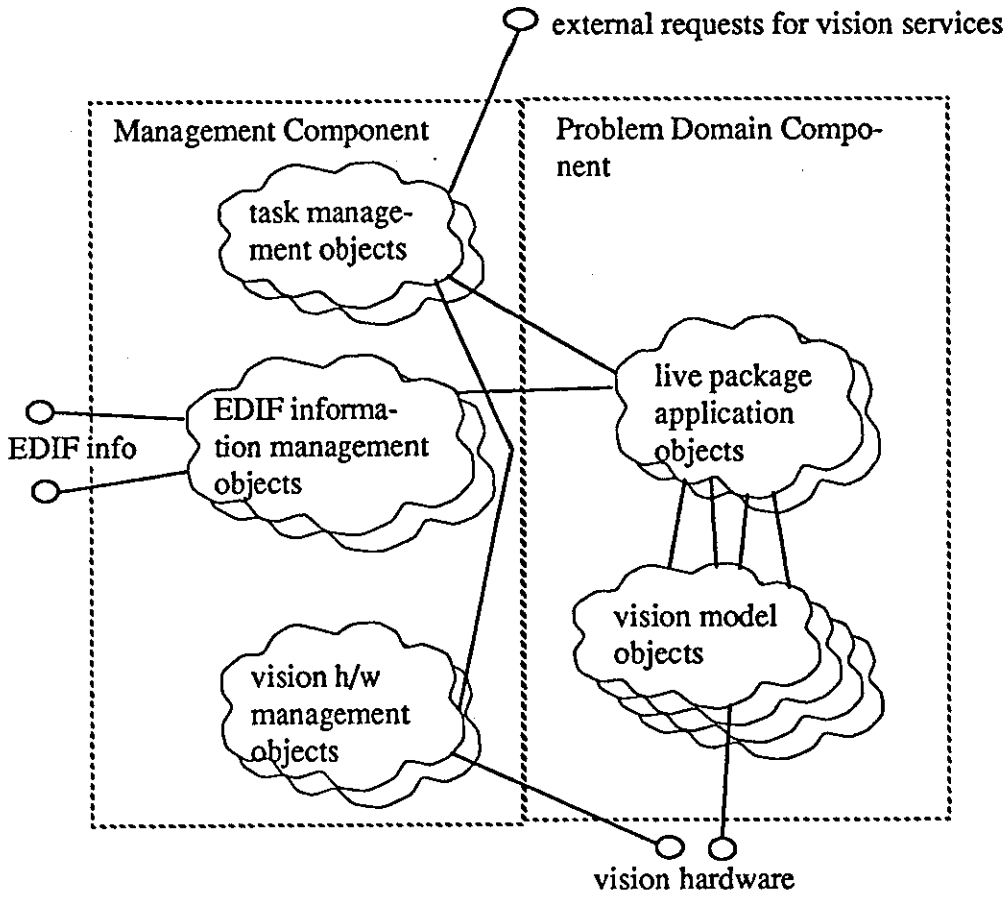


FIGURE 7. decomposition within the application layer



4.2 The Application Decomposition

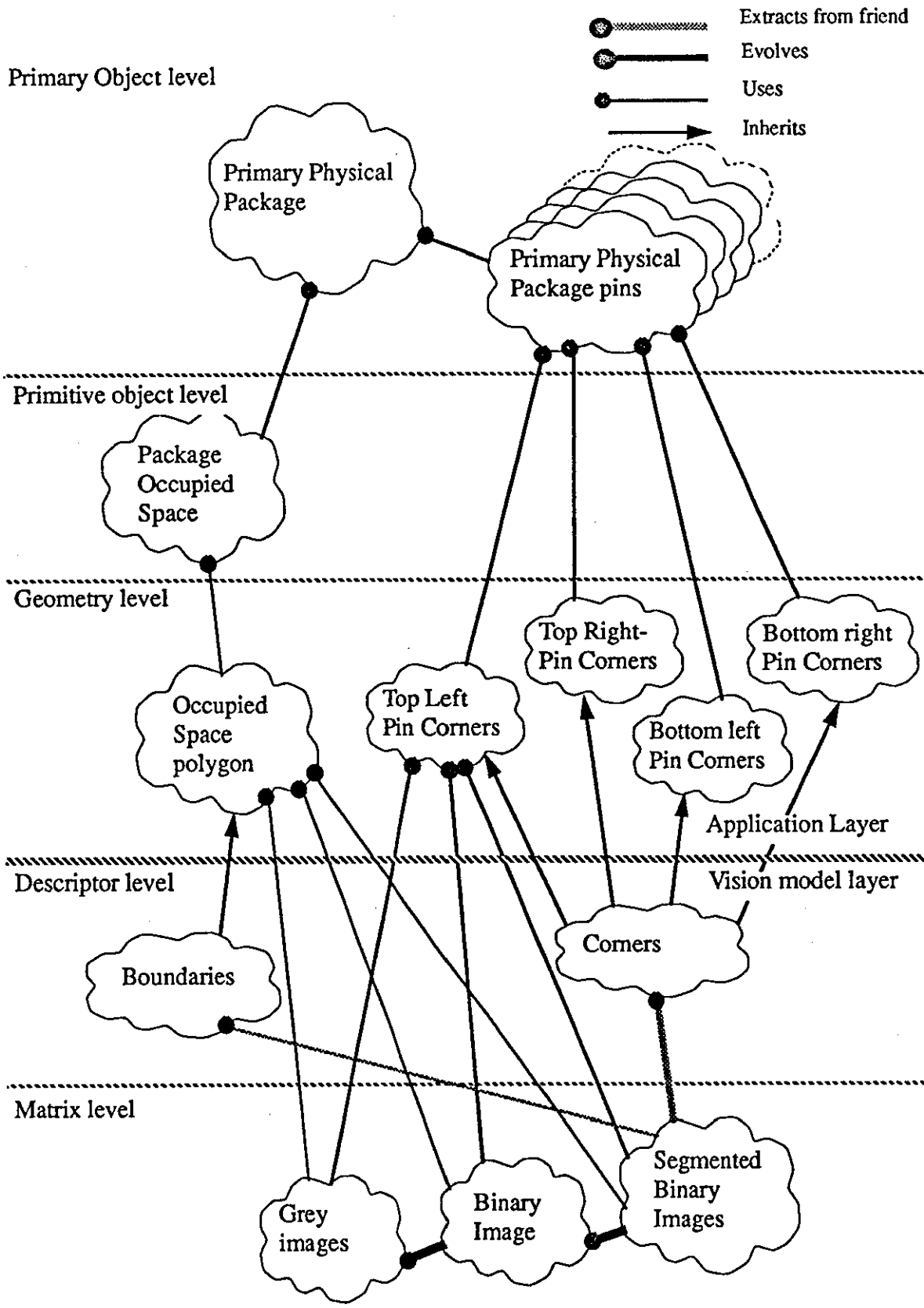
Figure 6 shows the object module diagram of the principal abstractions that are of interest to the designer responsible for the design and generation of the sub-system which will generate the live model information. The central object "the live component model information generator" is decomposed into two main components as described in Figure 7, these are the management component and the problem domain component. This decomposition is similar to that described by Coad [Coad 91] which identifies an emerging multi layer, multi component model common within object oriented software solutions. Reference to the Coad model and its parallels to this work is made in Appendix 4.

4.2.1 The Problem Domain Component

This component contains the application objects within a class hierarchy whose structure can be derived from the EDIF information model. The methods of these object classes are derived from the functional requirements of the application. In the demonstration application, the principal requirement of each object representing a physical entity is to generate a live description of itself. In addition, the other primary application-specific functional requirement is to provide some local component classification facilities. Figure 8 shows the Class diagram describing the decomposition of the EDIF-derived information abstractions. The object classes fall within the EDIF derived application layers of Primary Object, Primitive Object and Geometry Object.

Geometry Objects are derived from features extracted from the matrix images. In accordance with the rules governing the use of the vision model, Geometry Objects are derived classes, inheriting from descriptor level feature classes within the vision model.

FIGURE 8. Decomposition within the problem domain component - the Class diagram



This set of application-specific feature objects within the Geometry level forms a simple set of services which is offered to the bulk of the application functionality. In this application these comprise:

- "pack_ocspace_polygons" - primitive features representing the package occupied space;
- "top_left_pin_points" - a corner feature which can be used to represent a top left hand corner (typically of a pin or body of a component);
- "top_right_pin_points";
- "bottom_left_pin_points";
- "bottom_right_pin_points".

FIGURE 9. A typical object diagram

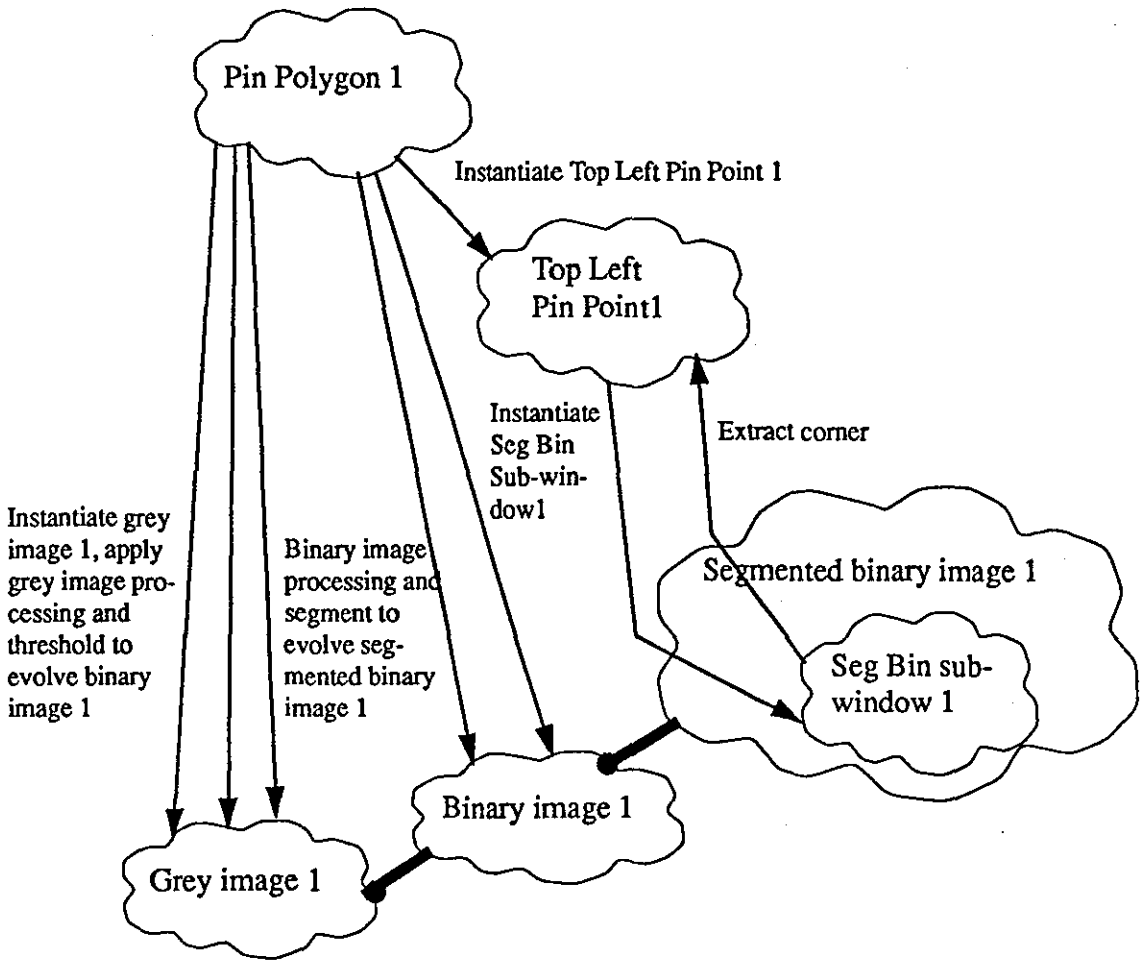


Figure 9 shows a typical Object Diagram which describes the instantiation and use of the vision model objects by the application objects. This is in line with the methods describing the decomposition of vision model and application issues proposed in chapters 5 and 6. The heter-

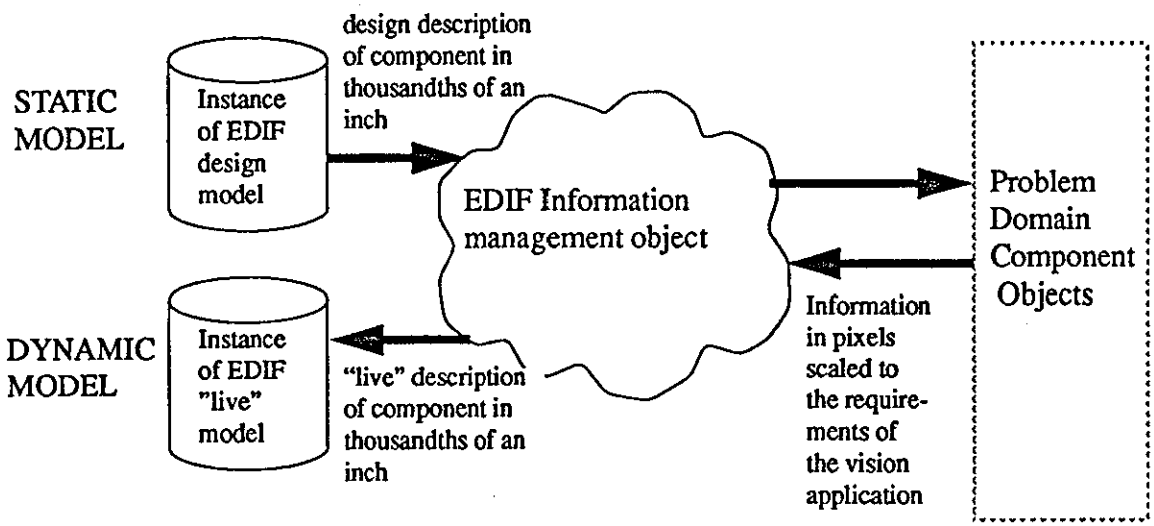
archical nature of the vision model allows the application objects the necessary access to the services for vision processing, provided by the grey and binary object classes. The "Top Left Pin Point" object (an instance of the derived class, "Top Left Pin Corners") uses the friend "Seg Bin Subwindow" object to gain access to the corner feature it requires.

4.2.2 The Management Component

The object classes identified within the management component, and shown in Figure 7, are application-specific and are derived by general analysis and design. The "Task Management" and "Hardware Management" objects are derived from the requirement for general control of the application and to initialise the vision hardware. The "EDIF information management" object provides a set of information management services to the problem domain component objects, as a form of information "Local View Provider". The concept of local views was introduced in chapter 6. In the vision application, what is required is a bidirectional view provision facility: as illustrated in Figure 10 and described as follows:

- specific elements of information from a neutral static EDIF model are converted to the form that is required to drive the vision application;
- live features derived from an image of a real electronic component are converted to neutral EDIF model form to populate a dynamic instance of the information model.

FIGURE 10. The EDIF information management object - a form of bi-direction information view provider



4.3 The Design Result

The services provided by the vision inspection application object and its support requirements are shown in Figure 11. The complete integrated vision system will access global information to enable component classification and could make use of the service request for elementary features. Functionality to enable local classification has been included to enable testing of the vision application object, and as a logical facility to improve speed, if the application already has knowledge of a particular component. In this case, local classification will first take place. If this fails the elementary features will be passed to the calling object. The calling object will be implemented as an open application running on an integrating infrastructure. As such it will be able to take advantage of the structured access to global information which can be used to implement a global component classification function.

FIGURE 11. The interface with the Inspection application object

- request elementary description for global classification
- request "live" model, attempting local classification

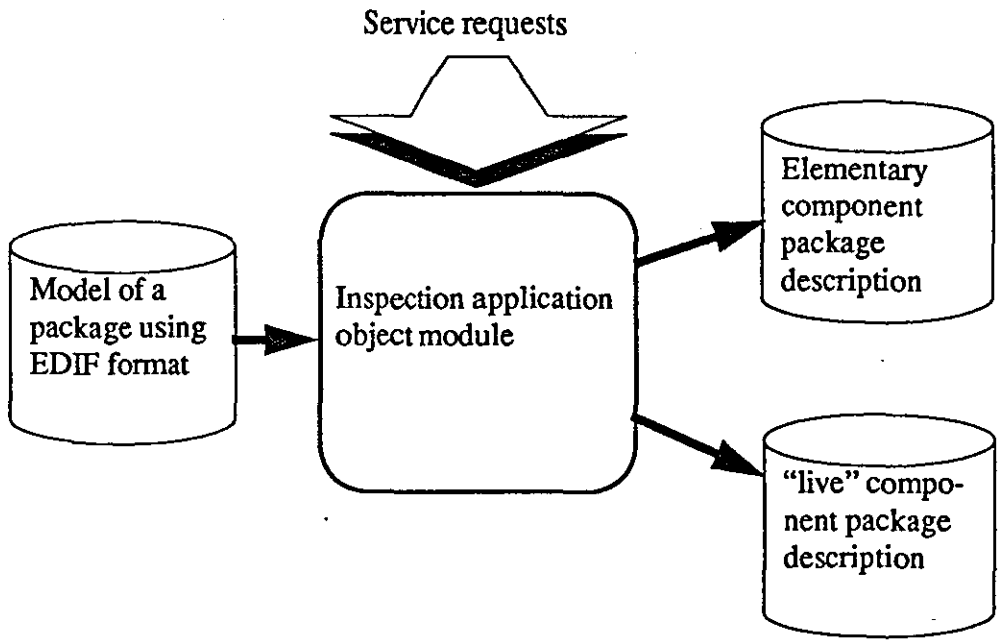
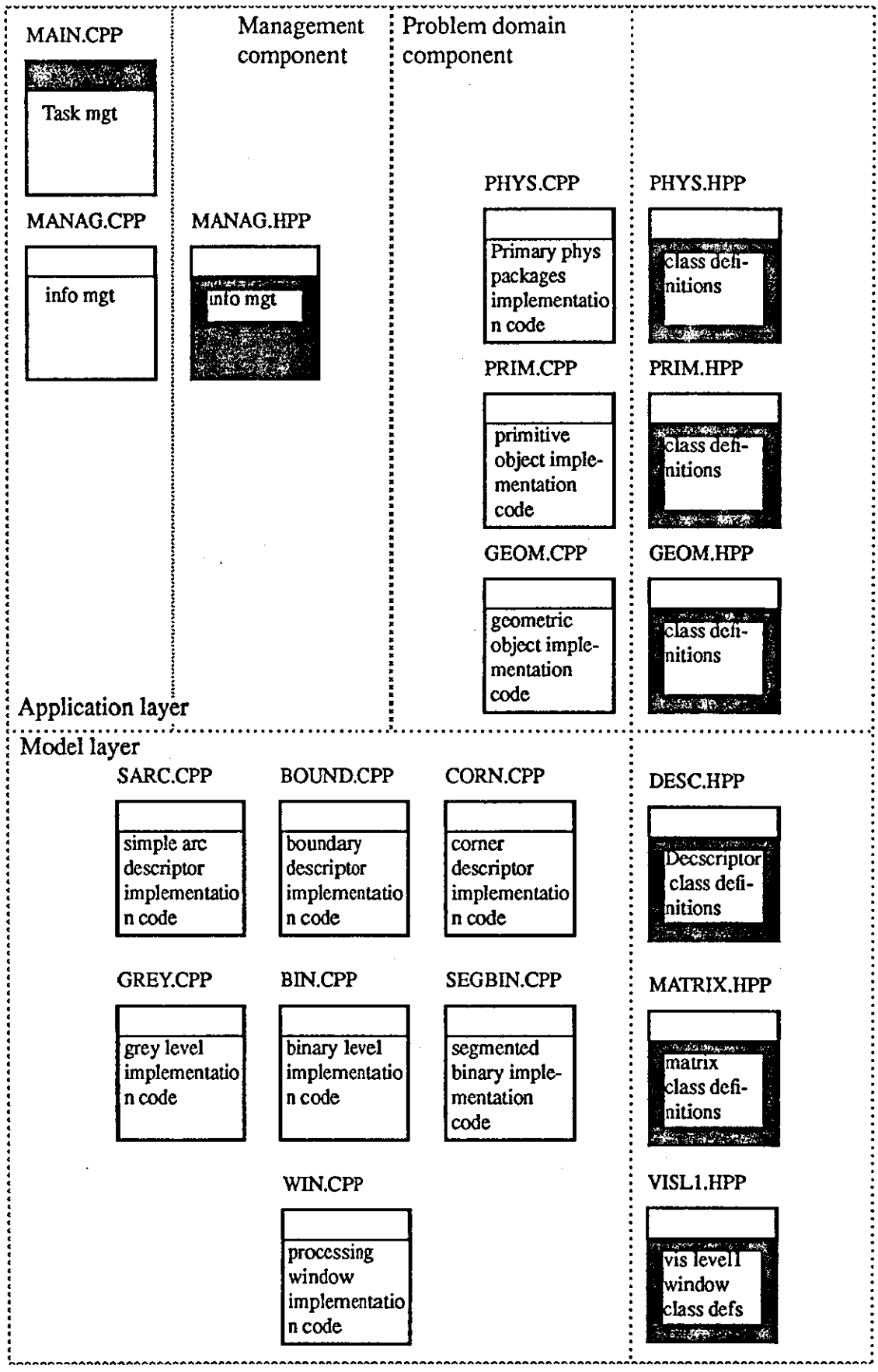


Figure 12 shows the breakdown of software modules used to implement the application object. This figure provides a further view which demonstrates the layered separation is carried through into implementation. The application during execution is described in Appendix 5.

FIGURE 12. Object sub-modules in the vision application object module



Chapter 8

A DEMONSTRATION AND EVALUATION OF THE MECHANISMS FOR HANDLING CHANGE WITHIN THE VISION APPLICATION OBJECT

1.0 INTRODUCTION

"The only Constant is change" [Bishop 89].

One of the principal themes of this thesis is the necessity for manufacturing systems to adapt to change. An ability to change constantly is necessary to ensure that manufacturing enterprises remain competitive in a global economy. The creation of a new generation of manufacturing applications which have the ability to facilitate change could contribute to the adaptability of the next generation of manufacturing systems.

The reference architecture described in this thesis includes mechanisms which will help next generation machine vision systems to adapt to change. These mechanisms are based on the following principles.

- The provision of structure through:
  - the use of a layered decomposition (within the vision application object) incorporating interfaces based on virtual machine abstractions [Seidewitz 86];
  - a domain specific decomposition within the vision application software based on the EDIF model for PCB;
  - a generic vision model embodying basic ideas of digital image processing and feature extraction.
- The use of diagramming techniques to support a design methodology based on Booch91 and the proposed vision model [Booch 91]. These techniques can support structured design and provide documentation that is an essential element in supporting change [Gillies 92].

- The use of structured software, where object orientated implementation (using C++) provides properties such as inheritance and polymorphism which can help support software change [Cox 87, Wiener 88].

This chapter discusses the problems of quantifying the degree of support provided by these proposals. The lack of established metrics to quantify the degree of improvement is identified, but a number of software qualities which support change are identified.

Three variables which affect the time taken to implement change are deduced from a discussion of contemporary work on software metrics:

- the degree of complexity of the required change;
- the degree of engineering resource employed in making the change;
- the degree of availability of support tools to enable change.

The chapter describes how two degrees of change were used to provide some feel for the relative improvement of the object-based system over earlier structured software. "Improvement" was measured qualitatively from the structure of the software and the reduced time to implement change

Evaluation of the proposals would be incomplete without considering cost. No quantitative investigation has been undertaken to establish the potential cost implications of using the proposed architecture. However, studies support the cost-effective use of a structured approach to manufacturing application design and implementation [Peck 87, Camp 76]. These are presented in Appendix 6.

2.0 HANDLING SOFTWARE CHANGE

During design and implementation, the incorporation of additional external services or internal functionality can usually be added to the original specification of a software system. Although some compromise to the ideal structure of the system may be necessary, change can usually be accommodated at this stage. Following the implementation of the complete system and its early use, which may expose missing elements in the original specification, further change may be necessary. This is usually considered an acceptable part of commissioning or

fine tuning of the system. It is also generally considered to be the most costly part of software development [McCall 77].

In the long term, change may be required in line with the key elements introduced in chapter 1, i.e. in response to a technology push or a market pull [Bishop 89]. Long term change in response to unforeseen requirements, cannot by definition be aided by built in functional capabilities designed to enable flexibility. This change relies on replacement or modification of hardware or software elements within the original system, and is supported only by the structural mechanisms used within the original design.

It is understood that the vision objects in this thesis are primitive, whereas it is the complexity of real systems which makes change a major problem for software systems. However we propose that this chapter and the final chapter of Section C demonstrate the advantages of the methods and mechanisms proposed. This chapter attempts to quantify the degree of support offered.

The previous chapters of this thesis rested on the assumption that the provision of the mechanisms proposed by the author can support the set of requirements identified in chapter 1. As an example it has been suggested that the additional structure, provided through compliance with the architecture, could support software maintenance (a common hypotheses [Gillies 92]). The following section discusses the problems of measuring the degree of support offered by such mechanisms.

3.0 SOFTWARE METRICS

3.1 Introduction

The term software metrics has come to refer to any activity within software engineering associated with quantification. Measures have been suggested for cost, productivity, complexity and software quality [Watts 87, Fenton 91, Gillies 92]. However, software metrics are still very much a fringe activity within mainstream software engineering [Fenton 91].

"Engineers from any other discipline are fully aware of the crucial role of measurement, and would be forgiven for assuming that it would form a central role in software engineering",

[Fenton 91] but Fenton states that within software engineering "nothing could be further from the truth".

3.2 Software Quality

Work on software metrics is based primarily around methods for measuring software quality. McCall proposed a model for software quality as early as 1977 [McCall 77]. This model is still the basis for recent work on measuring quality [Watts 87, Gillies 92]. The model comprises three areas, described by McCall as follows:

- Product Operation: requires that the software system can be learnt easily, operated efficiently and that the results are those required by the user.
- Product Revision: is concerned with error correction and adaptation of the system. This is important because it is generally considered to be the most costly part of software development.
- Product Transition: may not be so important in all applications. However, the move towards distributed processing and the rapid rate of change in hardware is likely to increase its importance.

It is McCall's third area that is of interest in this thesis. Product Transition was predicted by McCall in 1977 as being of potential significance due to the move towards distributed processing and the rapid rate of change in hardware [McCall 77]. Product transition embraces some of the qualities identified in chapter 1 as necessary for next generation vision machines i.e. the ability of a software system to adapt to change within an open distributed processing environment.

3.3 Measuring Software Quality

Figure 1 tabulates characteristics of the quality areas defined by McCall. This table is used by Watts (together with a similar model postulated by Boehm in 1978 [Boehm 78]) as a conceptual framework for measuring software quality. This figure identifies the following characteristics for Product Transition:

- portability - the effort required to transfer a program from one environment to another;
- reusability - the ease of reusing software in a different context;
- interoperability - the effort required to couple the system to another system.

FIGURE 1. A Quality Model based on McCall 77

USE	CHARACTERISTIC	ATTRIBUTE
Product operation	Usability	Operability Training/familiarisation Communicativeness I/O volume and rate Data commonality
	Security (integrity)	Control and audit of access Integrity of data, etc
	Efficiency	Storage efficiency Execution efficiency
	Correctness	Completeness Traceability Consistency
	Reliability	Accuracy Error tolerance Simplicity Consistency
Product revision	Maintainability	Consistency Simplicity Conciseness Self-descriptiveness Modularity
	Testability	Simplicity Instrumentation Self-descriptiveness Modularity
	Flexibility	Expandability Generality Self-descriptiveness Modularity
Product transition	Re-usability	Generality Self-descriptiveness Modularity Machine independence Software system independence
	Portability	Self-descriptiveness Modularity Machine independence Software system independence
	Interoperability	Modularity Communications commonality Data commonality

Figure taken from reference Watts 87

In his assessment of the work of McCall, Boehm and Watts, Gillies produces a table of available metrics against quality criteria [Gillies 92] (see Figure 2). Although a single metric is cited for portability, their are no current metrics for adaptability (an additional quality criteria identified by Gilb [Gilb 87]), interoperability and reusability.

Watts reports a number of approaches to measuring quality criteria. In principle these are more akin to the measurement techniques use in the social sciences rather than the physical sciences. Typically, a simple or weighted scoring system might be used to generate a score for each criteria, from which an overall mean score will define the total quality of the system.

Watts reports the use of a scoring system based on a polarity profile as shown in Figure 3 [Watts 87].

FIGURE 2. Metrics available for each quality criteria

Quality Criteria	Number of metrics cited
Maintainability	18
Reliability	12
Usability	4
Correctness	3
Integrity	1
Expandability	1
Portability	1
Efficiency	0
Adaptability	0
Interoperability	0
Reusability	0

Figure taken from reference Gillies 92

FIGURE 3. polarity profile for software complexity

extremely complex	-3
complex	-2
slightly complex	-1
neither complex nor simple	0
slightly simple	1
simple	2
extremely simple	3

Figure taken from reference Watts 87

The polarity profile shown in Figure 3 is used to measure software complexity. As with all the quality criteria a combination of attributes contribute to complexity. This creates the problem of defining reliable metrics e.g. lines of code, size constraint, speed constraint, number of sub-routines. McCabe's Complexity Measure, otherwise known as the Cyclomatic Number, is the commonly cited metric in this area [McCabe 76]. It is based on the number of independent paths through the code.

Gilb's work on measuring software quality was developed in parallel with that of McCall, Boehm, and Watts [Gilb 87]. Gilb proposed a more flexible approach to software quality measurement through the use of a template rather than a rigid model.

FIGURE 4. the GILB quality template

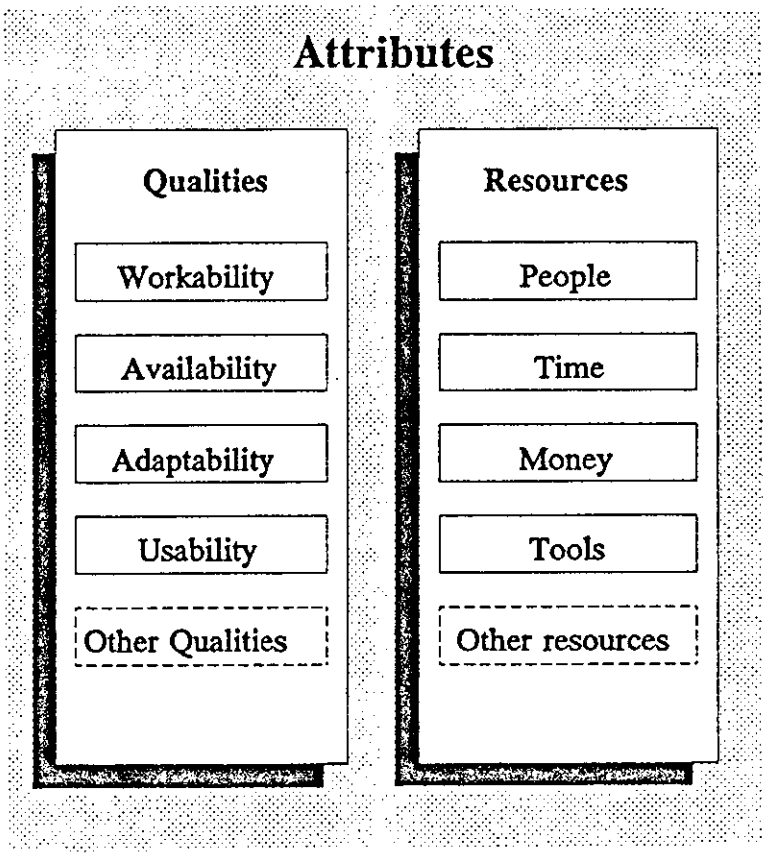


Figure taken from reference Gillies 92

3.4 A Template For Measuring Software Quality

The key feature of Gilb's 'quality template' is that it is designed to be tailored to local requirements [Gillies 92]. The underlying philosophy is that quality depends on a set of resources that vary from one application to another. It is the responsibility of the software engineer to identify the quality criteria that are important to a particular application. Quality is then built into the application in terms of the critical resources.

Gilb proposes four quality attributes and four resources as shown in Figure 4

FIGURE 5. Gilb's quality measures

Attribute	Sub-attribute	General Measure	Specific Example
Workability	Process capacity	Units per time	Transactions per sec
	Responsiveness	Action per time	Response time /secs
	Storage capacity	Units stored	Bytes per record
Availability	General	Probability available	Time available ÷ Total time
	Reliability	Mean time to failure	Total time ÷ Number of failures
	Maintainability	Mean time to repair	Time to fix 90% of test bugs
	Integrity	Wholeness	Degree of software intact
Adaptability	Improvability	Time for minor change	Time to add test set
	Extendability	Time to add a function	Time to add 10% logic
	Portability	Effort for transfer	Percentage of effort for porting
Usability	General	Degree of productivity	Time to reach basic level of ability
	Entry level	Qualification level	Readability
	Learning req'mt	Time to learn	Length of training required
	Handling	Net productivity	Tasks per hour
	Likability	Extent of positive attitude	Percentage surveyed

Figure taken from reference Gillies 92

In this case, adaptability is appropriate to this thesis. It may be considered in the following terms [Gillies 92, Gilb 87]:

- improvability - the time taken to make minor changes to the system;
- extendability - the ease of adding new functionality to a system;
- portability - the ease of moving a system from one environment to another.

The resource attributes in Figure 4 note that quality in any field cannot sensibly take place regardless of resource constraints such as cost [Eliott 93]. Figure 5 shows Gilb's measures for the attributes and subattributes defining software quality. Here Gilb uses the time taken to

make changes to software to measure the quality attributes of improvability and extendability. It is these two attributes that are of particular interest.

3.5 Conclusions

The work of McCall, Boehm and Watts has identified the lack of a metric for quantifying the degree of support for change a software system may possess. The work has illustrated the use of measurement systems for software that rely on generating a relative figure of merit. This is due to the lack of any simple, absolute and unambiguous scale typical of those to be found throughout the physical sciences [Gillies 92].

The first IEEE - issued standard on software metrics was approved in 1992, Software-Quality metrics methodology 1061 [Shneidewind 93]. The philosophy behind 1061 is that an organisation can use whichever metrics are most appropriate for its applications so long as the methodology is followed and the metrics are validated. The standard implies there is no specific measure of software quality which can be applied across the board, and supports Gilb's contention that a flexible approach is required. It should be noted that both the standard and the work of Gilb are contentious due to their inability to tackle the problem of generating a measure which can be used for comparing the quality of different software systems [Shneidewind 93, Gillies 92].

The object-orientation community is no better at providing software metrics. The recent books by both Gillies and Fenton [Gillies 92, Fenton 91] which provide detail of contemporary software metrics make no mention of object orientation. Bilow, in his guest editorial for the Journal of Object Oriented Programming, states that "object oriented software demands its own metrics, it is essential that the object oriented software community cultivate software measurement techniques" [Bilow 93].

In conclusion it is easy to see why Fenton contends that software metrics are such an immature discipline within software engineering. This situation makes it very difficult to quantify the benefits of the architecture for soft integrated machine vision and the implementation described within this thesis. The following is a proposal for relative measurement which draws from the work of both Gilb and Watts.

3.6 A Proposed Measure For The Support Of Change.

Gilb's quality attribute of adaptability included subattributes of improvability and extendability. Gilb suggests these could be measured through the time taken to effect a minor change and to effect a more complex change to provide additional functionality. This implies there is a variable describing the "degree of complexity of a required change". We believe the resources of people, money and time are not independent, and to a first approximation can be considered collectively as a second variable. This variable shall be known as the "degree of engineering resource available". This is based on the assumption that if they are financially justified, both people and time can be bought. However, it is understood that, under certain circumstances, time cannot be reduced linearly by increased engineering resource. Gilb's final resource attribute is tools. Here the author's architecture, models, methodology and mechanisms which can contribute to the extendability of a system can be considered to be "tools". The "degree of available support provided by tools" forms a third variable.

FIGURE 6. A proposed framework for quantification

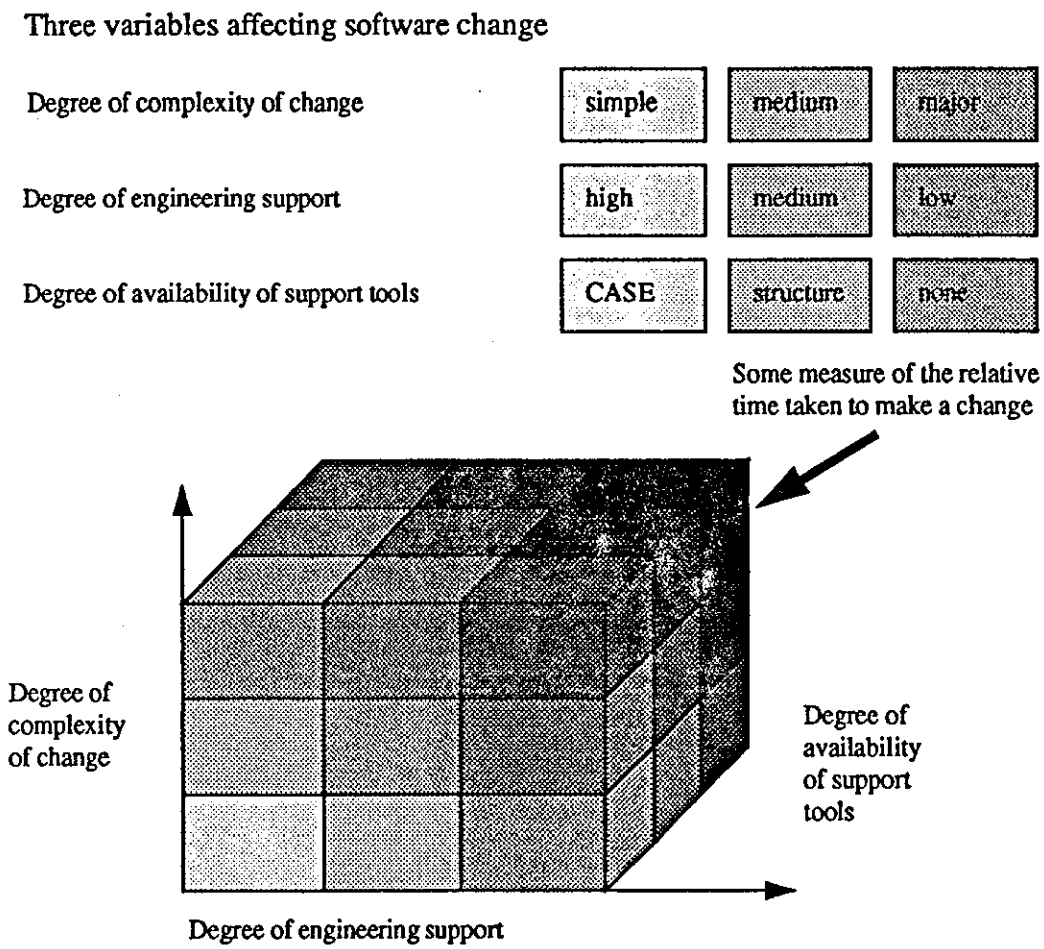


Figure 6 combines these variables in a cube whose axis are made up of the three variables. It can be seen in Figure 6 that the combination of low resource, few structured tools and a complex change requirement has the potential to create long timescales for completing a software change. Figure 6 provides a graphical representation of how the availability of support tools could contribute to reducing the time taken to make software change. By examining and modifying pieces of code a relative measure of the usefulness of some of the author's proposals can be made.

The code used is as follows:

- code generated by the author and described in the implementation in chapter's 7 and 8, this code adheres to the author's architecture;
- the author's early code to implement vision processing and feature extraction, as described in chapter 4, which obeys basic rules for structured coding and uses a layered architecture;

A measure for each piece of code is based on criteria identified for McCall's quality attribute of Product Transition and Gilb's quality attribute of Adaptability, plus additional criteria suggested by the author, namely:

- Modularity;
- Self descriptiveness;
- descriptive support
- machine independence;
- time taken to make a change.

A simple measure of complexity is also added to compensate for the complexity of the vision application object being much greater than that of the early experimental work. This measure is based on the number of bytes of executable code used in each case. This simple approximation is used to provide a relative feel for the complexity of each system.

Two examples of change are used, a simple change, and a more complex change to provide additional functionality. The two examples of change made to the vision application object are described in detail to demonstrate the mechanisms proposed by the author which support

change. Following these descriptions results obtained in regard to both types of change are presented and conclusions are drawn.

The following section describes a simple modification made to the vision application object. Within this section, the requirements for making a similar change to code produced during the author's earlier work is also briefly presented.

4.0 A SIMPLE MODIFICATION TO A VISION APPLICATION OBJECT

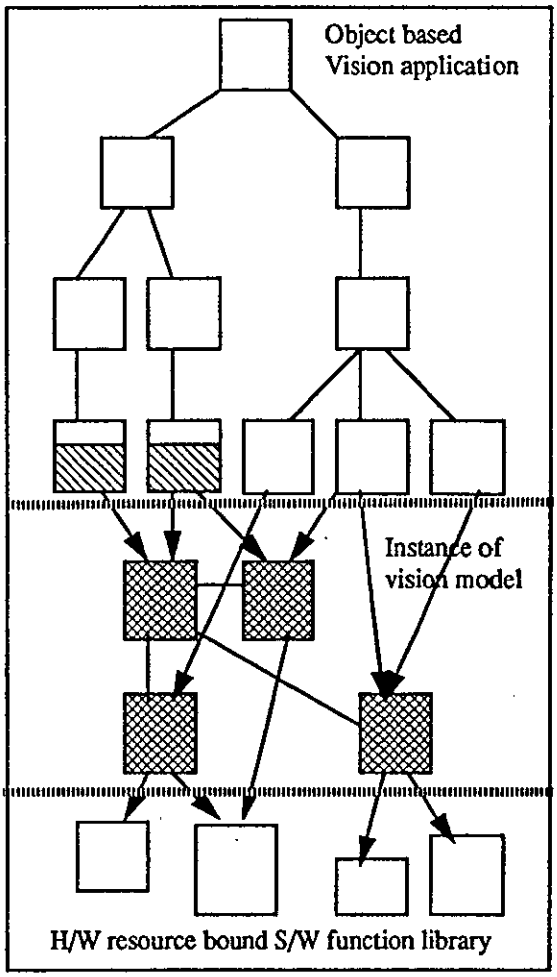
This example illustrates change to the required functionality of the vision application object. This could be due to a change in the component gripping technology needed to present components to the vision system camera. This change means that the component can no longer be illuminated from the rear, removing the advantage of a silhouette type raw image. Additional vision processing is required to ensure generation of a consistent binary image from the more complex raw grey level image captured from the front illuminated component.

Figure 7A shows a schematic representation of the modular structure of a vision application object, which adheres to the author's architecture. Part B of the figure shows the conventional hierarchical modular approach used in the first implementation.

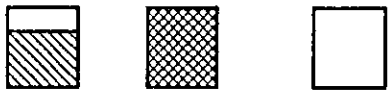
Part A of the figure shows the elements of the object-based architecture which are affected by the change. These elements can be immediately identified on the interface between the vision model implementation and the application layer. The vision model implementation may change but is simply another instance of the existing class library which implements the vision model.

Part B of Figure 7 identifies the problems which can occur when changing conventional software systems. A modification made to an existing function to satisfy a requirement for change can cause a "knock on" effect in other sections of code which require the same function in its original form

FIGURE 7. The modular structure of the vision processing applications



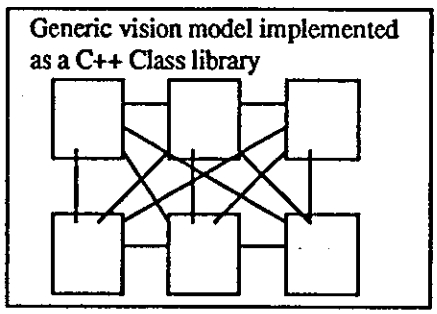
A) The proposed object based structure

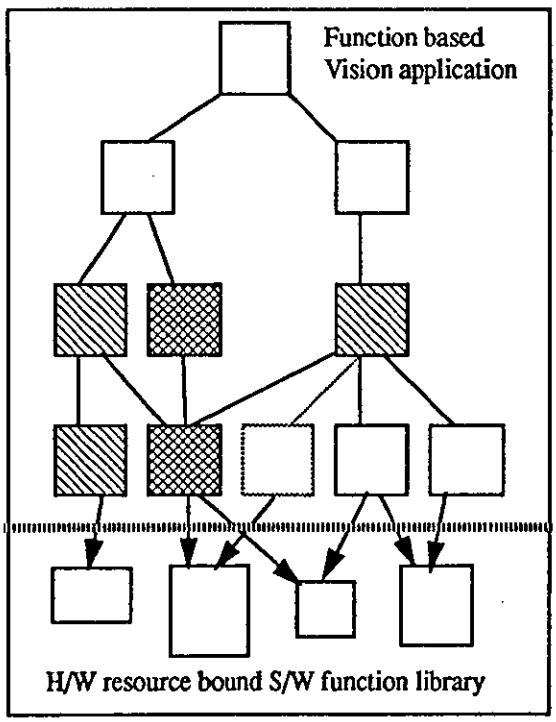


S/W change to object methods

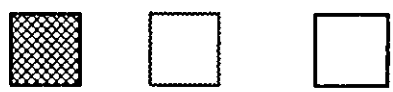
new instance of the vision model (requires no change to vision model implementation)

unchanged S/W





B) The early function based structure



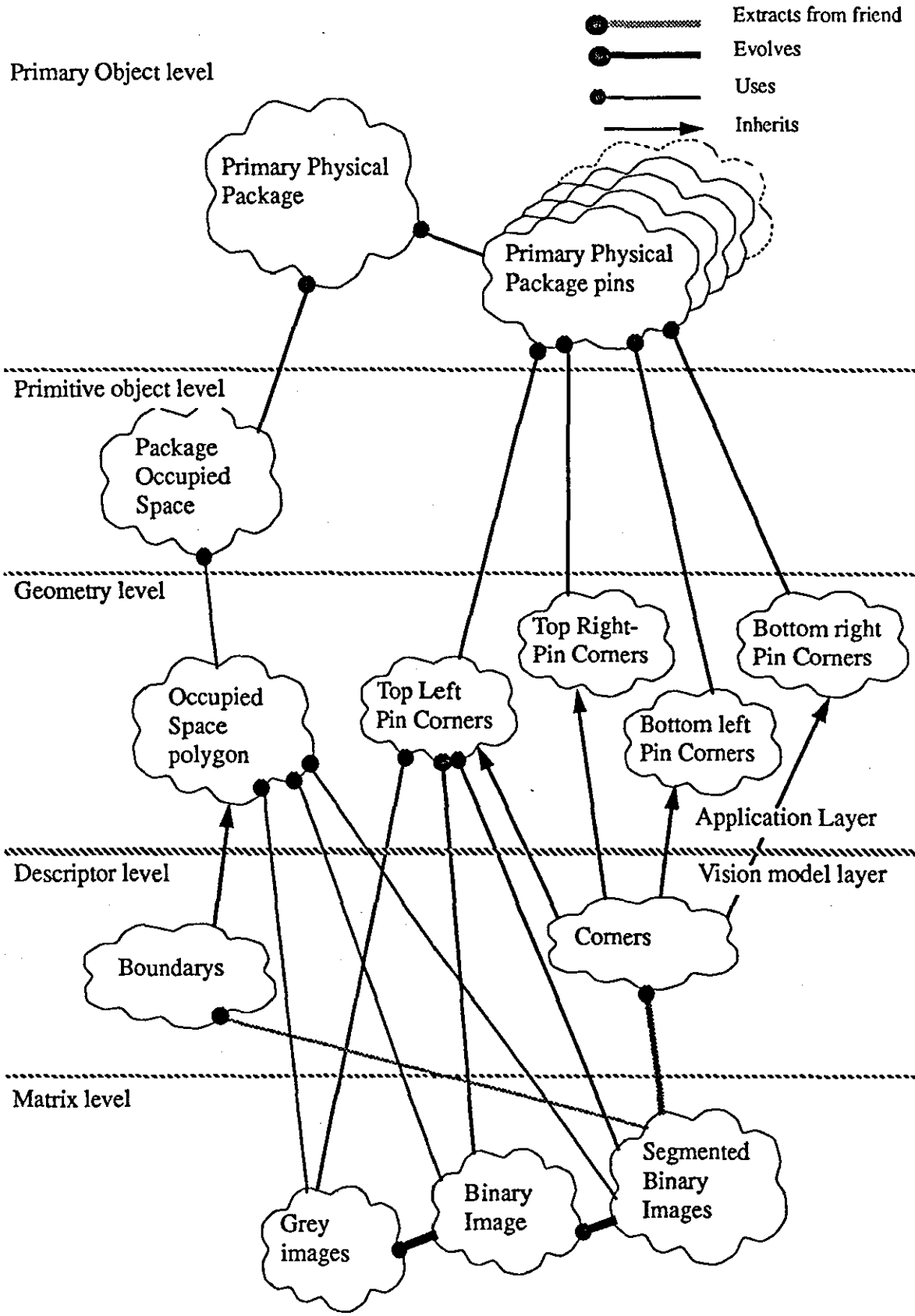
module requiring S/W change

new module required due to change to an original function

unchanged S/W

module requiring S/W change due to a "knock on" effect

FIGURE 8. Decomposition within the problem domain component - the Class diagram



The following observations can be made about the nature of the required change to the vision application object:

- no modification is required above the level of application specific feature extraction;
- no modification is required to the generic vision model (generic but constrained by the Matrox (H/W) and C++ (S/W));
- no additional features are required to be extracted.

The structure of the application object is such that all matrix level processing is contained within the application-specific object methods of the geometry level features (see Figure 7 and Figure 8). In theory, the modification can be achieved by providing the additional required vision processing within the geometry level object methods, such that their external behaviour remains the same, while their internal functionality has changed to cope with the new requirements. In practice the change is less straightforward.

A modification of this type illustrates what can and what cannot be achieved when making changes to image processing software. Figure 8 repeats the class diagram which shows an overview of the structure of both the vision model layer and the application layer software design. Assuming that the additional required vision processing algorithms exist as object class methods of grey and binary objects, no change should be required in the vision model layer. (If this is not the case, and the engineer implementing the change does not want to modify existing code, the existing grey and binary classes can be inherited and the additional algorithms can be implemented as methods of the new classes). If the vision model implementation is based on a comprehensive range of image processing algorithms all modification will take place within the applications layer.

The design of the vision model to applications layer interface and the C++ implementation detailed in chapter 7 is such that no application specific code exists within the model layer. All interaction between the two layers is through the application layer "looking into" the model layer and never the reverse. It is this concept that allows change to take place within the application while the vision model remains isolated and protected from the "knock on" effects associated with software modification.

The required modification to the application object is to compensate for a change in the captured image. This necessitates modification within the Geometry level (see Figure 8) so that the "Occupied Space Polygon" object instantiates a grey image and evolves a binary image. But its "use" of these objects is modified by invoking a different set of object methods which evolve a Segmented Binary Image which is as similar as possible to the original Segmented Binary Image. If this is achieved, the Boundaries Object can extract features similar to the original, and the rest of the code can remain unchanged.

The structured approach implemented through the use of the architecture, and through the use of the Booch-based diagramming techniques (to document both the design and the implementation), enables the engineer to modify the code in a structured way. A typical process is illustrated as follows:

- capture examples of the new raw image and use off-line image processing analysis to design the set of vision algorithms required to generate a consistent segmented image. The author uses Foster Findley PC-Image [Foster 90b];
- isolate the Occupied Space Polygon Class. This requires good documentation of the implementation modules i.e. the C++ code files and headers;
- write a simple test application to exercise the original Occupied Space Polygon Class. Create the associated Make files to structure compilation and linking;
- implement the required modifications through making additional calls to the vision model object methods within the Occupied Space Polygon constructor and its object methods;
- build and test the new Occupied Space Polygon Class using the test application;
- rebuild the complete application.

In this example, the engineer is using the hierarchy within the application which has defined interfaces, and so long as that interface is maintained, the functionality in each layer is immaterial. However, maintaining these interfaces is often difficult in image processing. In the case of the Occupied Space Polygon, it was possible through the use of a more complex thresholding routine and the use of morphological operations on the binary image. These operations generated a segmented binary image whose extracted boundary and associated features were

similar to that of the original. Appendix 5 shows photographs of the captured grey image and various stages of the matrix level processing during generation of the segmented binary image.

It proved impossible to generate an image from which meaningful descriptions of the Pin Corners could be extracted. The constraints imposed by the limited level of difference between important detail and confusing background within the captured grey image has implications which cannot be contained within the Matrix level of the Vision Model.

The early function-based code used as a basis for comparison in this exercise was described briefly in chapter 3. It is not ideal as a bench mark insofar as it is also structured, and adheres to a strict architecture devised during the early phase of the work. However, the comparison is interesting as it illustrates the benefits and draw backs of conventional C and object oriented C++ implementation.

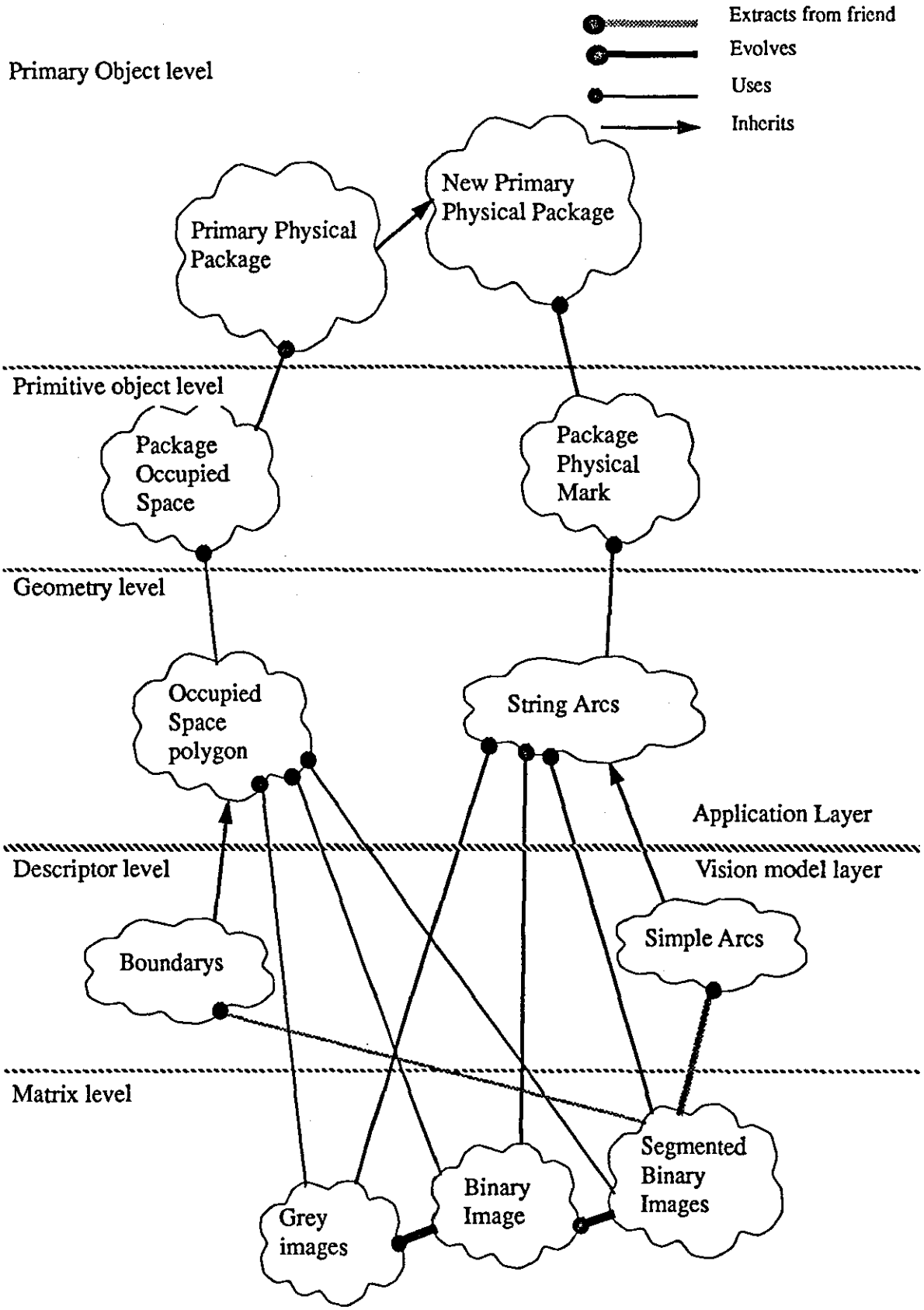
Figure 7B describes the structure of the code in general terms. Although it is based on the architecture described in chapter 3, the set of vision processing subroutines which call low-level library routines are application specific. If they are called by two separate functions at a higher level in the architecture a modification to the low level function (required by one of the high level functions) will cause "knock on" effects which could require the addition of new subroutines and modification further up the hierarchy.

5.0 A MODIFICATION TO PROVIDE ADDITIONAL FUNCTIONALITY TO THE VISION APPLICATION OBJECT

5.1 Introduction

A further change within a vision application object could be needed following the addition of a new open application within the soft integrated manufacturing cell or system. Such a change may require a new vision service. An example of such a change implemented by the author was to provide a vision service for a new client application. The application required the vision object to inspect the component manufacturer's identification code, which is typically a text string stamped on the top surface of each component (i.e. TI926X). The new application required a representation of the text based on a set of Arc features.

FIGURE 9. New decomposition within the problem domain component - the Class diagram



5.2 The Addition Of A New Vision Service Within The Vision Application Object

Two areas can be identified as requiring modification:

- a new set of features is required which describe the identification stamp. Image processing and feature extraction takes place through the generation of a new hierarchy of application objects within the application layer of the vision object;
- a new top level object is required which offers the new service but retains all original services;
- no modification is required within the vision model in the vision application object;

The overall design of the new service is based on the design principles proposed in this thesis. The EDIF Model for PCB [EDIF 90] is used to give a guide and naming convention to the objects identified within the application. The application architecture is that derived from the EDIF Model. The functionality of the service is based on the principle of extracting elementary features of a component and classifying the component under inspection (as used in the original application described in chapter 5). Knowledge of the classified component can then be used to generate a processing window which will ease the vision processing and feature extraction processes required.

The structured approach which adheres to the reference architecture, and uses the Booch-based diagramming techniques to document the original design and the implementation, enables the engineer to modify the code in a controlled way. Reference to Figure 8 shows the top level class of Primary_physical_package which "uses" Package_occupied_space. It is this branch of the class hierarchy that generates the primitive features for component classification.

The existing application must retain its original functionality, while the additional service is added. The following points identify the steps required in order to apply the change:

- examples of the new raw image are used with off-line image processing analysis tools to design the set of vision algorithms required to generate a consistent segmented image from which the new features can be extracted.

- the Primary_Physical_Package class in the Primary Object Level is "inherited" by the New_Primary_Physical_package class (see Figure 8 and Figure 9). This new class is tested to prove it performs all its original functionality through invocation of the class methods it inherits. In particular the new class can "use" Package_Occupied_Space to generate primitive features for component classification.
- the New_Primary_Physical_Package Class can now have its own class methods added which will generate of the new Arc features which describe the manufacturers identification code. The Object Oriented implementation through the use of C++ enables this additional functionality to be added without having to modify any of the existing applications layer Code. Figure 9 shows the new Booch Class diagram. The Primary_Physical_Package_Pins class and its hierarchy from Figure 8 have been left out to provide a clearer diagram.

5.3 Application Layer Design Based On The EDIF Model

Information entities which describe a Package are contained in the EDIF (EXPRESS) Information Model [EDIF 91] shown in Figure 10. The Package entity has attributes ID, which is an entity of type ID_STAMP. This is composed of a number of attributes typically describing the name and number which a component is given in the design of a PCB. No Package attribute describes the component manufacturer's ID which appears as a physical mark on the component. (As stated in chapter 7, the EDIF model has not yet addressed requirements for manufacture, having concentrated on design only). We propose an additional Package attribute Physical_Mark which would be an entity of type String, but could also be enlarged to handle non-text physical markings on components. It could then use the entities within the EDIF model which are used to described physical attributes of a PCB i.e. Line_Segment, Rectangle, Point etc.

Physical_mark would reside within the Primitive Object Layer of the EDIF-based layered decomposition and would use application specific features derived from the ARC features of the Vision Model. These appear as String_Arcs in Figure 9. Appendix 5 shows images of the vision processing involved in the new vision service.

Modification to the system which uses the object based structure has used the inheritance mechanism within C++ to get access to existing functionality which can be used by the new

service. It would be very difficult to implement a major modification, such as the addition of a new service, to software which was modelled on the early function based structure. It is more likely that the new vision service would be provided through the generation of an entirely new piece of software to ensure the original code remains fully functional. The new modules of code may use some of the image processing functions shown in Figure 7B as a means of accessing the low level vision library. However, there are no mechanisms within the C implementation for inheriting large sections of useful code from the original implementation.

FIGURE 10. The EXPRESS information model of the EDIF representation of PACKAGES

```
ENTITY package;
        has_pins : SET [0 : #] OF pin;
        id : INTERNAL ID_STAMP;
        acronym : OPTIONAL INTERNAL JEDEC_ACRONYM;
        package_class : INTERNAL PACKAGE_CLASS;
        body_volume : INTERNAL THREE_D_SPEC;
        occupied_space : INTERNAL THREE_D_SPEC;
        true_shape : OPTIONAL INTERNAL SET [1 : #] OF THREE_D_SPEC;
WHERE
        only_occurs_in_relation_to :
            valid_users (PACKAGE,
                         ['PACKAGED_PART.IS_PACKAGED_BY',
                         'MOUNTABLE_PACKAGE.USES_PACKAGE']);

        can_exist_alone :
            can_exist_alone (PACKAGE, TRUE);
END_ENTITY;
```

6.0 RESULTS AND CONCLUSIONS

The table shown in Figure 11 lists the approximate time taken to make the changes described in this chapter.The table also shows the criteria for adaptable software identified in section 3.6. By examining the object based and function based code it is possible to identify the features of the software which could contribute to providing qualities which support these criteria. These qualities have been listed in the table in Figure 11.

FIGURE 11. comparison of the object based s/w and early function based s/w

OBJECT BASED STRUCTURE	FUNCTION BASED STRUCTURE
MODULARITY	
C++ object class library based on vision model	data driven subroutines at all levels within the system
application objects based in domain specific model	
three layered architecture within the vision object	
vision application object	
layered architecture within interoperation and interaction across the integrating infrastructure	
high level application objects in a distributed vision machine	
SELF DESCRIPTIVENESS	
C++ implementation promotes self descriptiveness through object method calls i.e. grey_image_1 evolve binary image 2	use of appropriate subroutine names and addition textural comments
use of naming convention derived from objects in the application domain i.e. the EDIF model	
DESCRIPTIVE SUPPORT	
use of Booch based diagraming techniques	no formal documentation
MACHINE INDEPENDENCE	
use of Zortech C++ which adheres to ANSI standard	use of Microsoft C V5.1, does not adhere to ANSI but was the defacto standard C compiler.
COMPLEXITY	
total number of bytes of the executable code used to implement vision processing and feature extraction	total number of bytes of the executable code used to implement vision processing and feature extraction
21.6 K bytes	11.0 K bytes
APPROXIMATE TIME TAKEN TO MAKE A CHANGE(HRS)	
A simple change 3	3
A major change to image processing 10	15

6.1 Supporting Change

The first 4 measures (based on the Watts approach) show the object based system to have many more attributes which support the criteria of adaptability. No specific figure can be put on how much more support for adaptability is offered by the proposals, but significant improvement is clear.

When applying Gilb's measure of time taken to effect a simple change, the systems can be modified equally easily. This dichotomy between measured adaptability and time taken to effect a simple change can be partially attributed to the greater complexity of the object based system, but has more to do with the use of C++. With a simple change, the advantages of an object oriented implementation i.e. inheritance of large sections of code, may not be appropriate. A well-structured C program can have a clarity which makes it easy to understand relative to a C++ implementation. Although more coding is required to implement the change to the C program the time taken was approximately the same.

When making a major change, adding new functionality to the vision application object, the time taken to modify the object based code was significantly less than the function based code. Although the object based code was more complex, major parts of the original code were required for the new function i.e. the code required to generate the basic features of the I.C. under inspection. The inheritance mechanism of the object oriented implementation allowed the use of this code without the problems which can be caused by modifying original code. The additional documentation provided by the Booch-based Class and Object diagrams provided knowledge of the overall structure of the system. The domain specific EDIF based architecture provided a structure for the new image processing design. In contrast the modification to the function based system was done through a new code module developed from scratch.

On the downside, C++ is difficult to master. All software written on the Sun workstations is implemented in C. Even after a number of years using both systems, C can feel a more comfortable programming language. It is important to point out however that this may in part be attributed to the lack of sophisticated C++ debug support tools available to the SI Group.

Any architectural framework, methodology or implementation mechanism needs to be understood by the design and implementation engineer who will make use of it. The time penalty involved with this learning exercise also represents a cost which must be considered. This cost aspect is discussed in Appendix 6.

In general, structured techniques for the design and implementation of general purpose software systems are of increasing importance as systems become more complex. This implies

that complex machine vision systems could derive greater benefit from the architecture than that demonstrated in this chapter.

6.2 Fulfilling The Needs Of Next Generation Machine Vision Systems

Chapter 1 classified the criteria for next generation machine vision systems as either application criteria or platform criteria. The proof of concept implementation described in PART B demonstrates that the application criteria can be fulfilled using the author's reference architecture. The following subsections summarise how the reference architecture (or tools), used to build the implementation, help to fulfil the platform criteria.

6.2.1 Easing The Design And Build Of Machine Vision Applications

The use of a systematic approach to the design and build of machine vision applications is demonstrated, based on the following mechanisms:

- the availability of a generic vision model, describing image processing and feature extraction, implemented as a C++ class library (see chapter 5 and 7);
- the use of a domain specific information model (EDIF) to guide application design through
  - identification of layered structure, and
  - identification of relevant objects (see chapter 6 and 7);
- the provision of structure, within a vision application object, by a three layered architecture. This decomposes vision application issues, generic properties of image processing and feature extraction, and issues pertaining to the image processing resource used in the system implementation. (see chapter 6 and 7);

6.2.2 Easing The Modification Of Machine Vision Applications

The improved support for machine vision software change derives from:

- increased modularity as structural mechanisms ease design (see previous subsection);
- increased self descriptiveness by using C++. (see chapters 5,6, 7 and 8)
- provision of descriptive support using the Booch-based design methodology to provide graphical documentation (see chapters 5, 7 and 8);
- use of the inheritance mechanism within the object orientated paradigm;

- the use of virtual machine interfaces at layer to layer boundaries within the architecture (see chapter 6, and 8);

6.2.3 The Use Of A Heterogeneous Range Of Vision Resources

As well as identifying how vision processing hardware heterogeneity can be supported by the three layered architecture, the work exposes the limits of the approach used, and proposes requirements for true heterogeneity. The following points summarise these conclusions:

- conceptually, object methods can be removed from the application software and be replaced by object methods that call upon different vision hardware (see Chapter 6);
- the constraint within the mechanism above is the requirement that the software interface to the hardware must be compatible with the software used to implement the vision model and application layer (see Chapter 6);
- greater heterogeneity of vision hardware might be achieved with an agreed international standard that specifies a consistent form for the software which provides an interface to vendor specific vision hardware. (Such standardisation would provide increased scope for the kind of portable image processing algorithms library (IPAL) suggested by Carter et.al. [Carter].)

6.3 The Usefulness Of Object Orientation (OO)

The following points summarise the usefulness of object orientation for the design, implementation and maintenance of machine vision systems as described in PART B.

- **General:** The general OO concept of associating data abstractions with a set of appropriate methods, which control the manipulation of their contents, has been shown to map well onto image processing and feature extraction. Grey level and binary images and their associated processing algorithms are established classifications within vision technology. Extracted features which could make appropriate Base Object Classes are also common across the industry e.g. edges, corners, regions and texture primitives etc.
- **Friend relationships:** Because of the necessity to "extract" features from the data which is private to a segmented binary object, the strict philosophy of OO must be broken through the use of the "friend relationship" available in C++.

- **Evolution:** The evolutionary relationship between matrix level object classes required complex implementation code due to the lack of support for such a relationship within the OO paradigm. A construct based on selective evolutionary inheritance would be useful in this domain.
- **Inheritance:** The use of inheritance enabled existing code to remain undisturbed while additional services which make use of this code are added. The usefulness of inheritance is dependent on the careful selection of objects within the original application. The author's proposals for identifying objects in the application domain support the design of appropriate object classes which have a potential for change.
- **Sequential and concurrent processes:** Examples demonstrating the benefits of object oriented systems usually use concurrent processes, where a set of related objects react in response to some external stimuli. This scenario is suited to the concept of private data abstractions controlled via methods that are invoked through messages passed between objects. Machine vision systems are essentially made up of a set of sequential operations where information is transformed from some complex abstraction to some simple abstraction. This can cause additional complexity in the implemented code.
- **Implementation:** It has been shown that the complexity referred to in the previous point can lead to simple systems, or systems requiring simple changes, deriving limited benefit from an OO implementation. For example, with a simple change the advantages of the inheritance of large sections of code, may not be appropriate.

PART C

THE FLEXIBLE INTEGRATION OF MACHINE VISION APPLICATION OBJECTS WITHIN MANUFACTURING SYSTEMS

Chapter 9

A PROPOSED ARCHITECTURE FOR SOFT INTEGRATED MACHINE VISION

1.0 INTRODUCTION

The vision Application Object described in PART B embodies many features which fulfil the requirements of a new generation of soft integrated machine vision systems. The external behaviour of this object provides vision services. In order that this object can form part of a useful open integrated manufacturing system, where it can offer vision services to open client applications, the requirements for its implementation within an integrated manufacturing system must be established.

This chapter proposes a framework for structuring the software required to link client applications with the services provided by the vision application object. This model, together with proposed implementation mechanisms, provide an integration methodology which makes a further contribution to the objective of the thesis, namely providing support for systematised implementation and change, within an open distributed system.

The framework is based on elements which address the following issues:

- the architecture of soft CIM building blocks which can be plugged into and removed from an infrastructure which underpins a CIM system. This involves the separation of the following three issues:
  - manufacturing application functionality;
  - application interoperation functionality;
  - application interaction functionality;

- the provision of interaction mechanisms which involve the use of services provided by an integrating infrastructure, and the buffering of alien devices such that they become compliant with the integrating infrastructure;

- the structuring of interoperation mechanisms which involve the creation of a virtual vision server and corresponding support for client applications, through a set of vision service functions. The virtual vision server then requires a mapping onto the real vision application object.

Proposals are made through identifying requirements for a soft integrated vision machine, implemented on the CIM-BIOSYS integrating infrastructure. The architectural elements of soft CIM building block are defined.

2.0 BUILDING BLOCKS OF SOFT INTEGRATED MANUFACTURING SYSTEMS

2.1 Current Thinking Within The OSI Community

The essential purpose of OSI is to support distributed processing [MacKinnon 90, Pimentel 90] i.e the inter-working of two or more application objects. Application layer 7 of the OSI model is the layer at which OSI services are made available to inter-working application objects. Figure 1 [MacKinnon 90] depicts the relationship between applications and the OSI environment (OSIE). MacKinnon explains how "Application Processes" (which refer to open manufacturing applications or Building Blocks of integrated systems within this thesis) can be viewed as operating in two environments. One is the local system environment, which performs the application object functionality, the other is the OSI environment which is "the union of all communications functionality related to the distributed application" [MacKinnon 90]. MacKinnon stresses that within a real Open System there is no explicit boundary between the two environments, and the separation is conceptual. Mackinnon also identifies the part of the application process within the application layer that deals with communication. He terms it the "application entity", as detailed in Figure 1. Within this thesis a MacKinnon "application entity" could be described more specifically as an "application object interoperation entity" that will embrace both the issues of application interaction and of application object interoperation. The remaining part of the MacKinnon application process which embraces the discrete application functionality is termed the "Application layer User", equivalent to the Application Objects described within this thesis.

FIGURE 1. Local System and Open System Interconnection environments

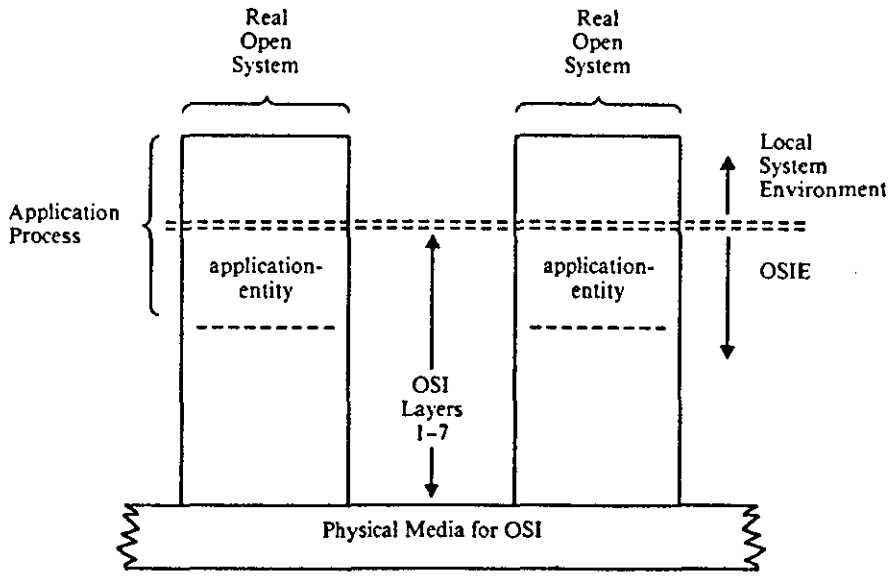


Figure taken from figure 7.1 of reference MacKinnon 90

This description of Open Systems Integration is analogous to Hard Integration, where issues of application interaction and discrete application functionality, though conceptually separate, are implemented as a single entity.

Inter-working of application objects is supported within OSI layer 7 by the specification of a coherent set of communications functions termed "application service elements" (ASE's). ISO is currently developing a number of ASEs, which typically include file transfer, virtual terminal, job transfer, distributed databases and document transfer [Pimentel 90, MacKinnon 90]. In general, an application layer protocol comprises a combination of these ASE's. Figure 2 provides a representation of an Application Process (AP) made up of an Application Layer User (ALU) which makes use of services offered by a layered set of ASE's.

Within the domain of Open Integrated Systems, an integrating infrastructure such as CIM-BIOSYS is positioned at a level similar to that of an ASE. The CIM-BIOSYS platform of services is directly equivalent to an ASE while its configuration and management facilities are analogous to the proposals for mechanisms to support Open Distributed Processing [Brenner 87, MacKinnon 90]. Continuing the relative positioning of CIM-BIOSYS within the OSI model, the CIM-BIOSYS communications drivers are loosely based on the OSI model layers

1 to 6. CIM-BIOSYS is a tool for integration, in its role as an ASE within OSI Layer 7 it offers managed integration services to applications. It can be used in combination with other ASE's or to embrace existing ASE's. Most importantly, the additional configuration and management facilities which make up CIM-BIOSYS and underpin its integration services enable Open Systems interaction to be implemented in a soft integrated manner.

FIGURE 2. View of the application service provider

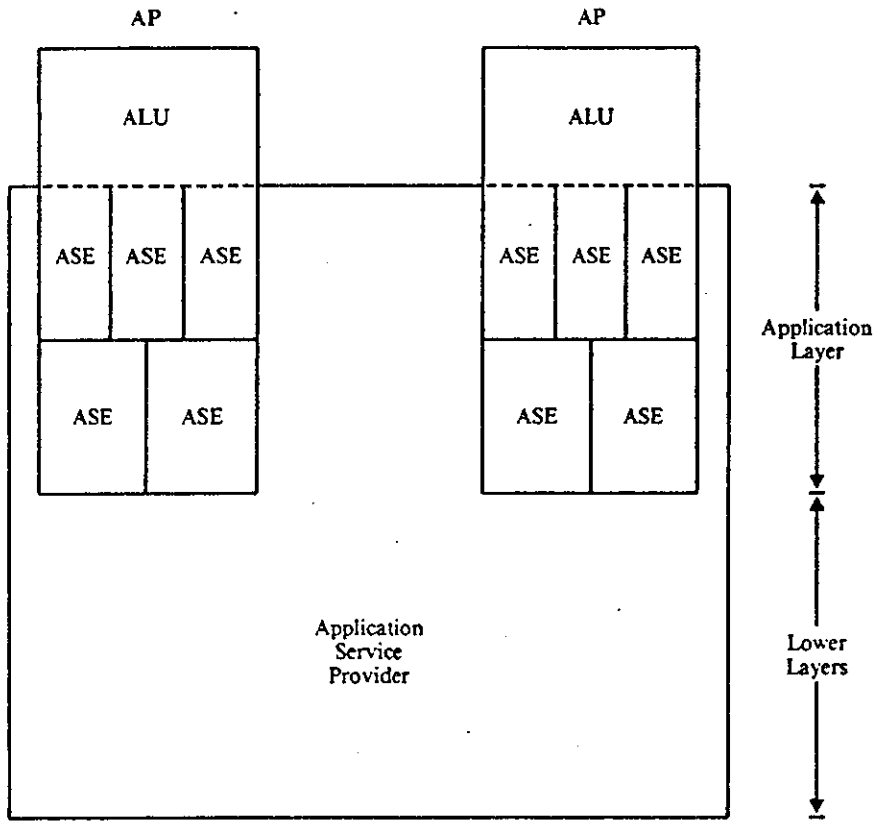
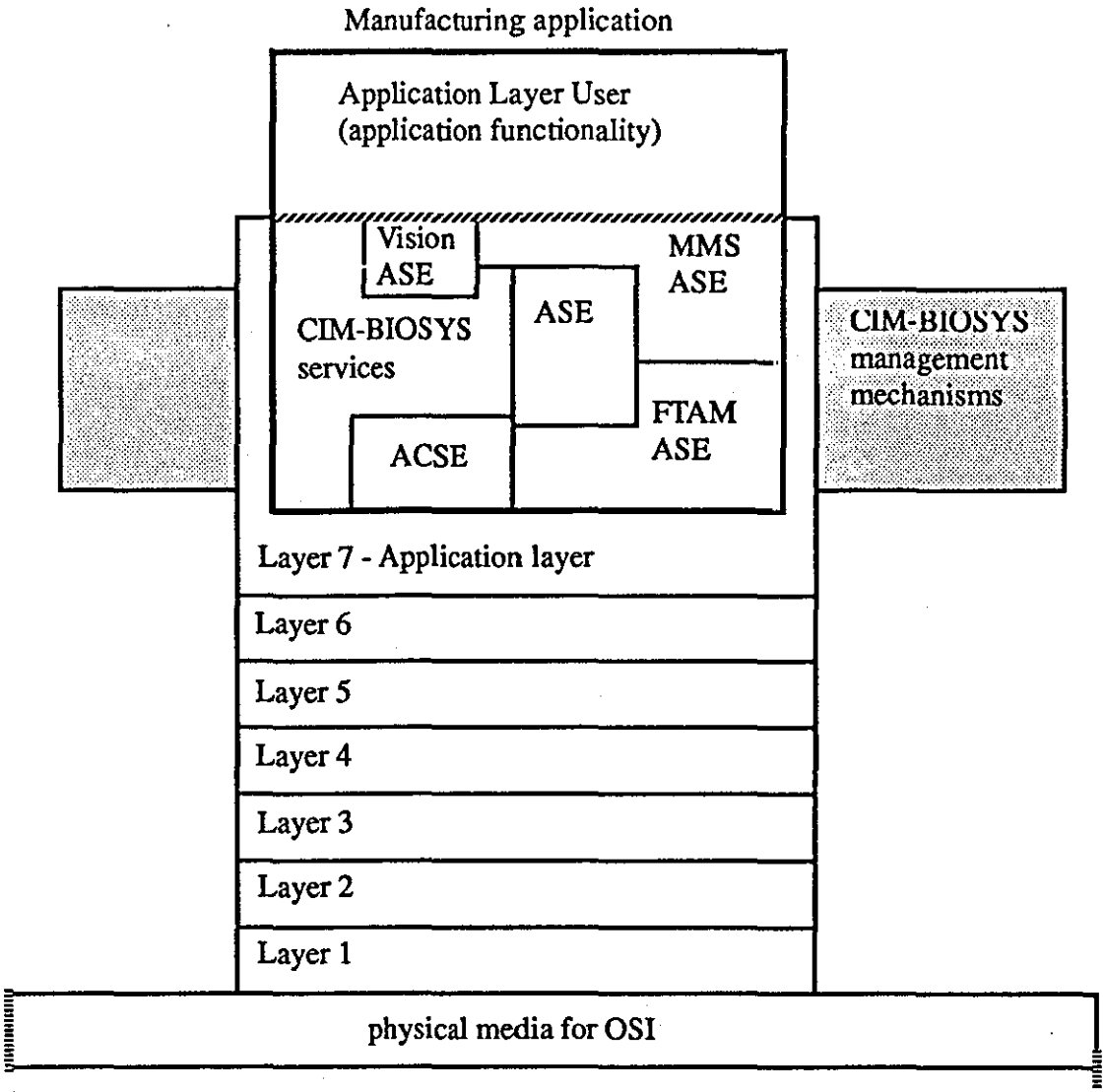


Figure taken from figure 7.2 of reference MacKinnon 90

We propose that further mechanisms are required to carry the "soft" integration philosophy through to support application interoperation. These mechanisms should support the separation of application issues and interoperation issues which are considered to be conceptually separate [MacKinnon 90] but are currently implemented with no explicit boundary. The author recognises the need for an application service element (ASE) to support the interoperation of vision processing resources / services and client manufacturing applications within an open distributed system. Figure 3 shows CIM-BIOSYS services, the author's Vision ASE, and other ASE's within the context of OSI.

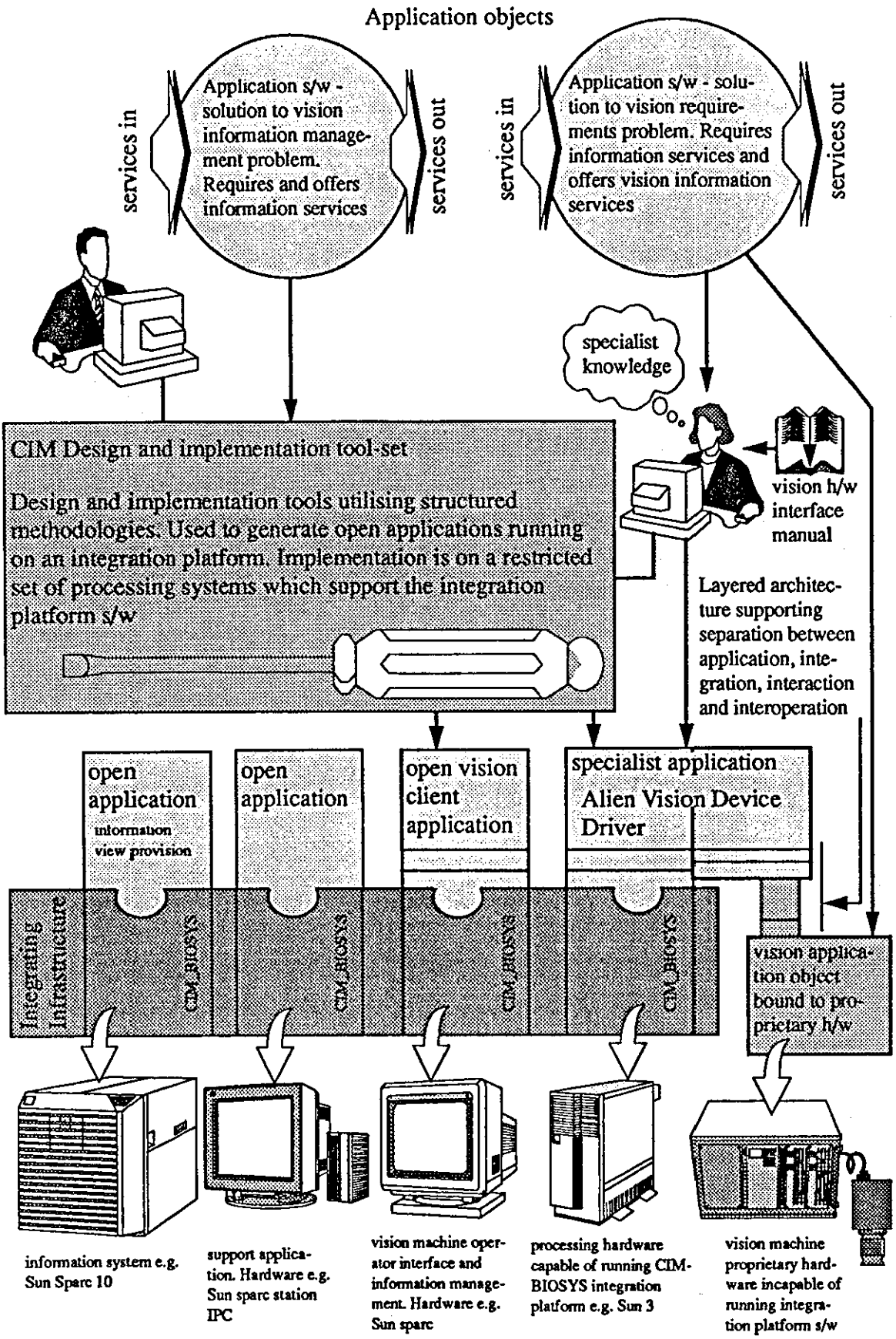
FIGURE 3. CIM-BIOSYS with respect to OSI and layer 7 application services



2.2 A Soft Integrated Manufacturing System Based On An Integrating Infrastructure

Figure 4 repeats Figure 6 in chapter 1, which describes a form of implementation when mapping application software on to a soft CIM system. The CIM system is constrained by the hardware and software used to implement the components of that system. The lower part of the figure describes a scenario where the CIM-BIOSYS integrating infrastructure is available and running on a set of hardware and software. An inspection system made up of distributed components, aggregated within the soft CIM system includes an application object. This object provides vision processing services, implemented on a specialist remote processor. This processor is incapable of supporting the integrating infrastructure.

FIGURE 4. The design and implementation of a present soft integrated vision system



The upper part of the figure shows two application objects. The left hand object represents an open application typically comprising operator interface and information management functions. This object requires general purpose processing resource and can thus be mapped onto a CIM-BIOSYS compliant host system during CIM systems design and implementation. The right hand object in the figure represents the vision processing application whose design and implementation was described in PART B. This object requires the resources provided by the specialist remote vision processor which is a CIM-BIOSYS alien device.

In order to generate a soft integrated manufacturing system where both application objects appear as discrete open applications (or soft CIM building blocks) within an integrated whole, two specific issues must be addressed:

- the remote vision processor or CIM-BIOSYS alien device must be buffered in some way such that it appears to the CIM-BIOSYS software as a compliant device (i.e. provision of a CIM-BIOSYS alien device driver);
- the application objects must adhere to some predetermined messaging protocol such that they understand each other's messaging dialogue. If they are to be implemented as applications providing open interoperation they must be supported by specialist mechanisms (as proposed within this chapter) and/or adhere to some recognised messaging standard.

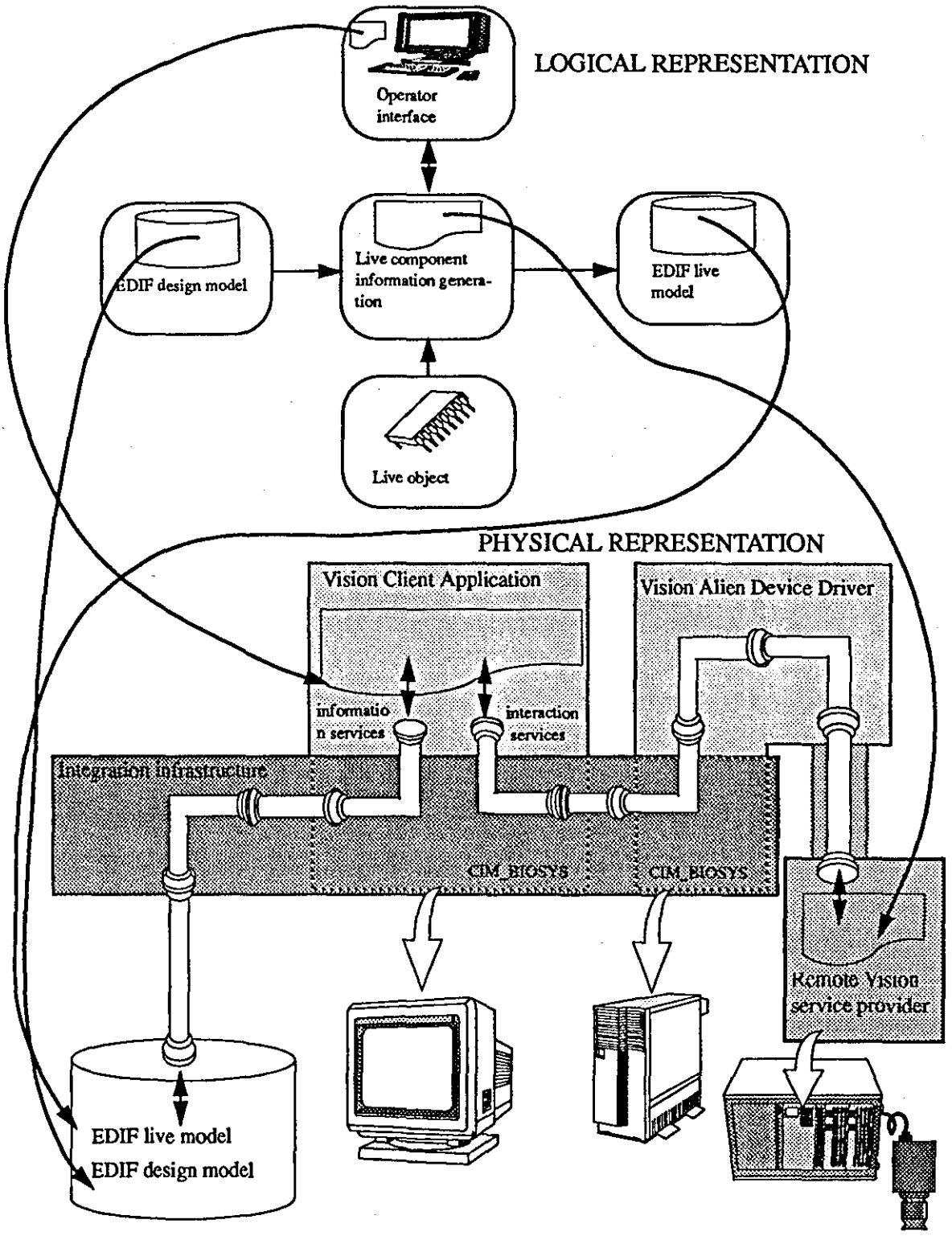
2.3 A Soft Integrated Vision Machine Implemented On CIM-BIOSYS

Figure 5 shows a mapping of the system object modules introduced in PART B onto the CIM-BIOSYS integrating infrastructure shown in Figure 4. This figure illustrates how the complete logical solution for the model driven inspection application can be implemented on CIM-BIOSYS. The "operator interface and information management" object is implemented as a CIM-BIOSYS compliant open manufacturing application. The "live component information generation" object is implemented as a vision application object on a CIM-BIOSYS alien device. It is the mechanisms required within the relationship between these two objects which are the primary issues reported within this chapter.

Figure 6 identifies the principal elements relating to the interworking of a CIM-BIOSYS compliant manufacturing application and the vision application object on the alien device. The figure positions the two specific areas of interest introduced in the previous section - those issues

pertaining to CIM-BIOSYS compliancy (i.e. object interaction), and those which are concerned with application interoperation. This figure also identifies the author's proposals for the necessary elements for an application to be considered a Soft CIM Building Block.

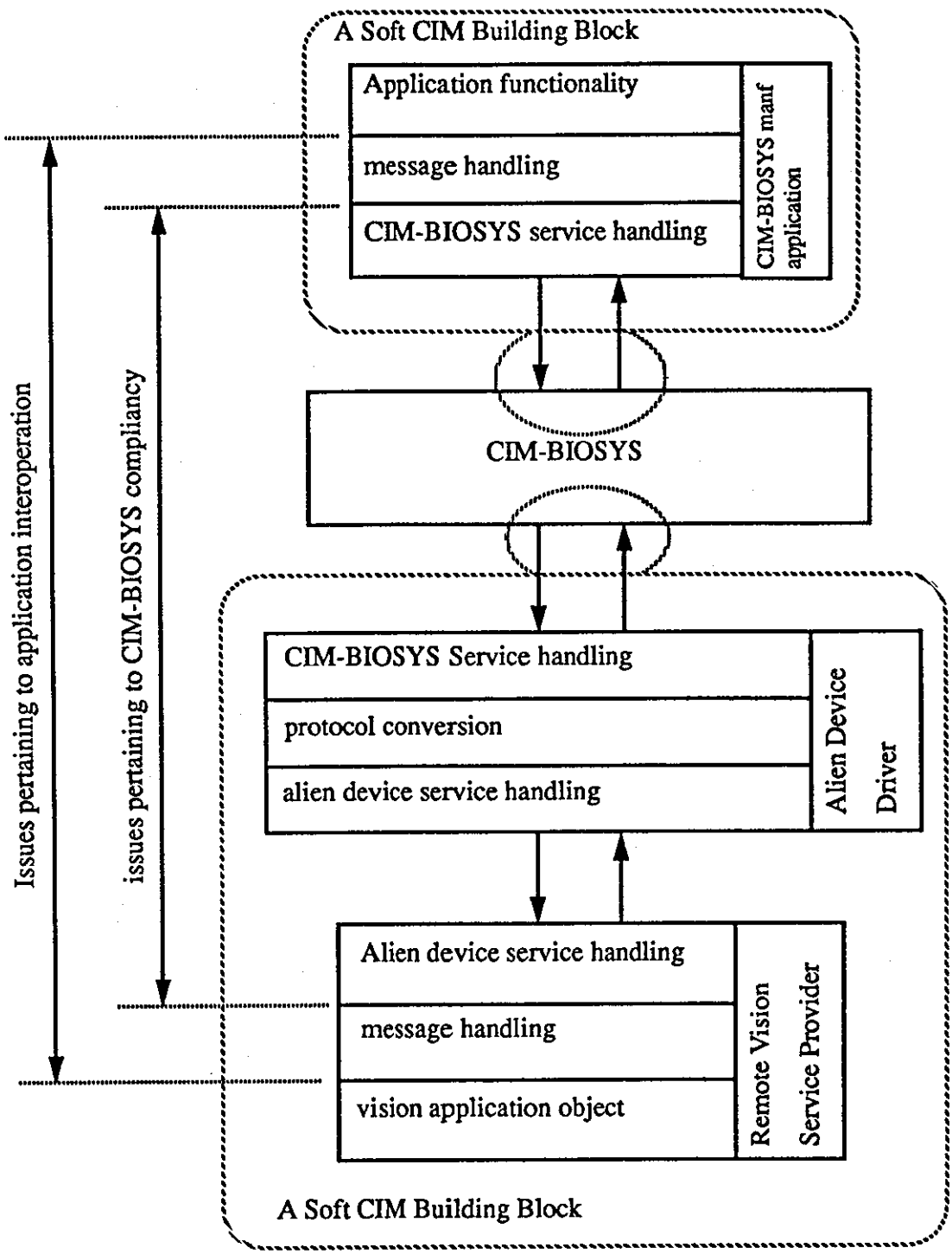
FIGURE 5. Mapping of the logical requirements onto the physical implementation



These elements comprise the following:

- manufacturing application functionality;
- application interoperation functionality;
- application interaction functionality;

FIGURE 6. principal elements within the interaction of a manufacturing application and the vision application object, identifying elements that make up a Soft CIM building block



3.0 PROPOSALS FOR SUPPORT OF APPLICATION INTERACTION

Structured and managed application interaction can be supported through compliance with the services of an Integrating Infrastructure. CIM-BIOSYS interaction compliancy requires that two or more interacting applications communicate by invoking and responding to CIM-BIOSYS application services (see Figure 6).

The version of CIM-BIOSYS available to the author provides two sets of services:

- application interaction services providing a mechanism for the transfer of data packets between compliant applications;
- information services, providing access to information stored in logical file format. (recent system enhancements have provided much improved information service provision based on configurable information objects mapped to a heterogeneous range of databases via information view provision facilities [Clements 91,92]).

Simple application interaction only requires the following four services:

- Establish Link: forms an association between two applications;
- Terminate Link: releases the association between two applications;
- Request Status: is used by an application to interrogate another application about its status in terms of associations with other applications, and;
- Send Data: which is used to send a data packet to another application.

To provide CIM-BIOSYS compliancy, enabling application interaction, applications should be able to invoke the four interaction services and handle incoming CIM-BIOSYS data packets. Figure 6 shows the requirement for a "CIM-BIOSYS Service Handling" module in a CIM-BIOSYS compliant manufacturing application. Figure 6 also shows the additional requirement of an Alien Device Driver to provide CIM-BIOSYS compliancy for the Remote Vision Service Provider which contains the Vision Application Object.

4.0 PROPOSALS FOR SUPPORT OF APPLICATION INTEROPERATION

The Manufacturing Message Specification (MMS) [MMS 90a. MMS 90b] defines the meaning and structure of a set of messages which can be used to control, monitor and pass information between programmable devices, but does not define the way data is transmitted or the nature of the network technology. It provides an independent layer which can provide an internationally agreed standard interface to enable discrete application objects implemented on programmable manufacturing devices to interoperate.

A key concept in MMS is the ability, by use of MMS services, to create Virtual Manufacturing Devices (VMD). A VMD exists within a server application and constitutes that part of the application that makes available a set of communications resources and provides a consistent range of services to client applications. The use of a VMD as a virtual interface for interoperation enables a range of different variants of a class of manufacturing device to interoperate with client applications, such that all variants of the device now appear to be uniform. The VMD is another mechanism which supports application interoperation, by using an abstract representation of specific functionality provided by a real manufacturing machine [Pimentel 90]. This virtual to real relationship is shown in Figure 7, as is the Client / Server relationship.

Figure 8 describes a similar set of relationships for a soft integrated vision machine running on the CIM-BIOSYS integrating infrastructure. This figure positions the CIM-BIOSYS integrating infrastructure (illustrating its platform of services) and the CIM-BIOSYS service user within both the client application and the alien device driver. The vision service user, or vision ASE, within the client device is shown, as is the vision server, or virtual vision machine, within the alien device. The virtual vision machine maps open services onto the real vision application object.

FIGURE 7. MMS Virtual Manufacturing Device within the client server model

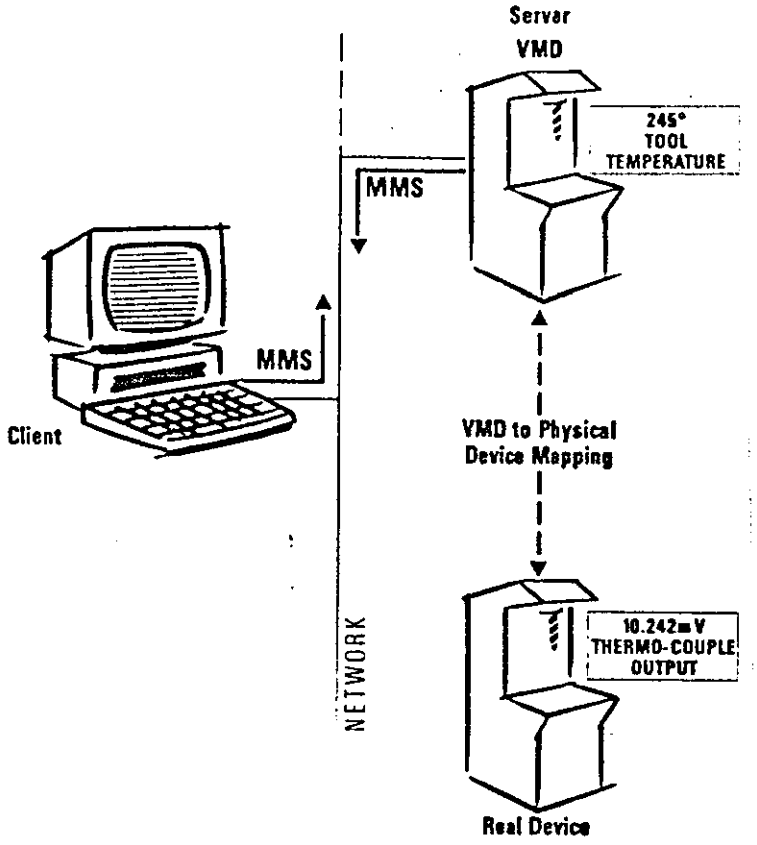
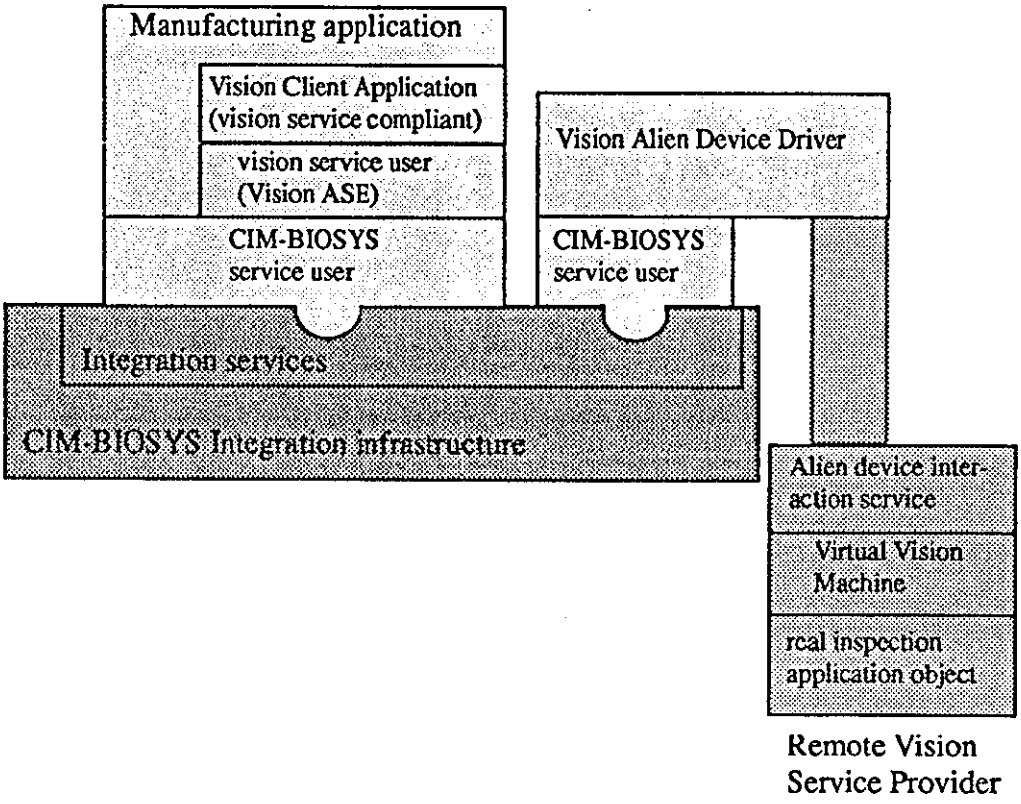


FIGURE 8. The vision client / server system and the Virtual vision machine

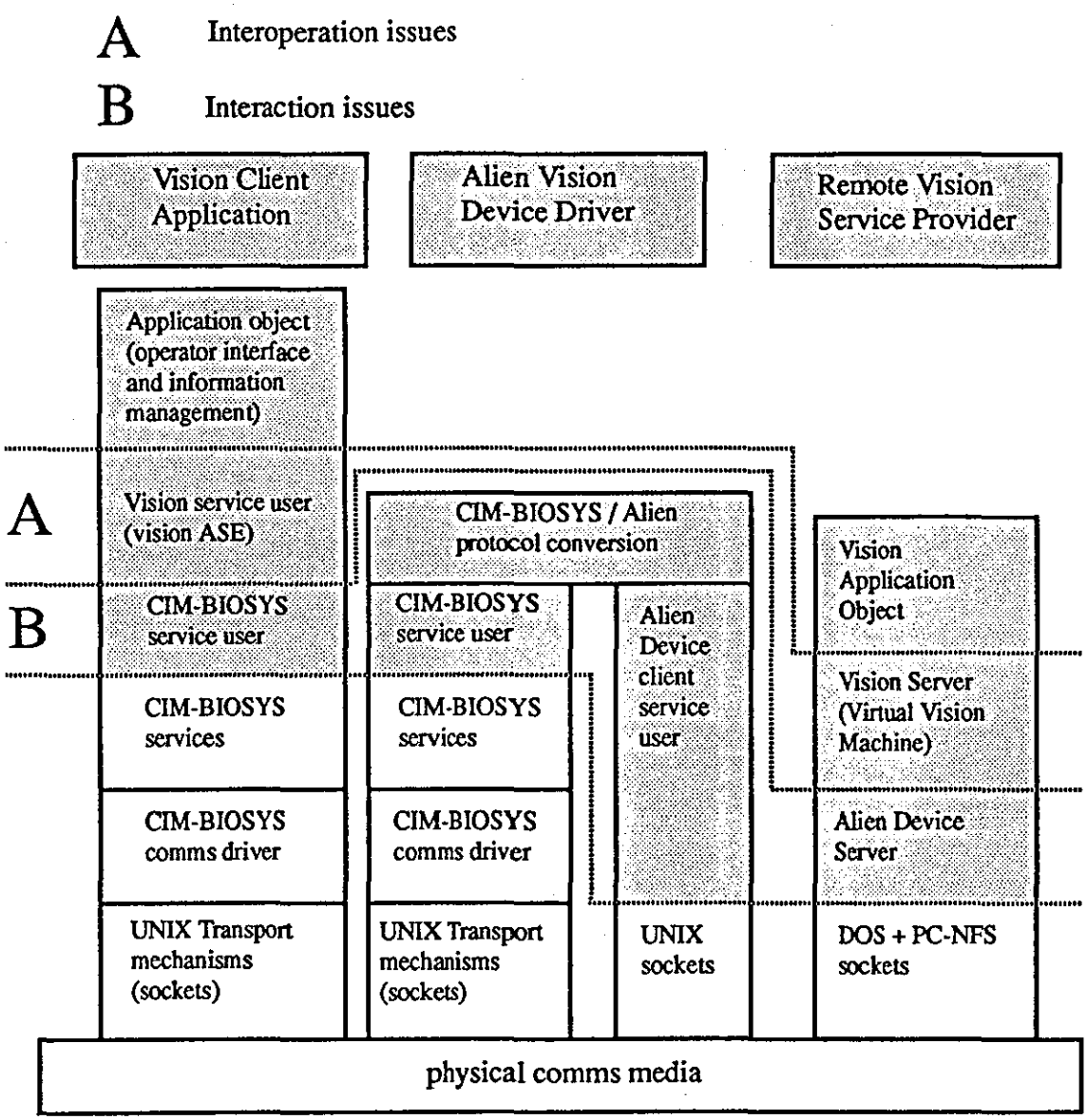


5.0 AN ARCHITECTURE FOR SOFT INTEGRATED MACHINE VISION SYSTEMS

Figure 9 combines elements of Figure 6 and Figure 8 to provide a layer by layer decomposition which make up the elements of the author's proposed architecture for a soft integrated vision machine. The decomposition between application object interaction issues and interoperation issues is clearly indicated, as are three discrete processes i.e. The Vision Client Application, the Vision Alien Device Driver and the Remote Vision Service Provider.

The following chapter details the mechanisms required to implement the elements of the author's proposals, and demonstrates their support for change.

FIGURE 9. Elements within the Soft Integrated Vision System



Chapter 10

IMPLEMENTATION MECHANISMS WITHIN THE INTEGRATION ARCHITECTURE, AND A DEMONSTRATION AND EVALUATION OF THE SUPPORT FOR HANDLING CHANGE

1.0 INTRODUCTION

This chapter describes the implementation mechanisms used to build a proof of concept soft integrated machine vision system based on the proposals detailed in the previous chapter. The first two sections comprise the following:

- a description of the elements used in the author's implementation, to provide application interaction via the CIM-BIOSYS integrating infrastructure;
- a description of the mechanisms used to implement application interoperation. This interoperation enables the discrete application objects described in PART B of this thesis to work together to form a soft integrated vision machine.

The implementation described in this chapter is in line with intermediate or next generation open systems. Specialist facilities to enable interaction between open applications and the vision application object are needed because specialist vision hardware is needed which cannot run CIM-BIOSYS software. The generation of a vision alien device driver for CIM-BIOSYS is thus necessitated. The significance of this implementation is not only in the mechanisms described for interaction and interoperation but also because it offers a migration path from current vision machine solutions, i.e. discrete devices, to future fully open distributed vision applications. These fully open vision machines would be based on resources that can support a common integrating infrastructure.

The architecture for integration described within PART C includes mechanisms which will support the requirement of next generation integrated machine vision systems to adapt to change. These mechanisms are based on the following principles.

- The provision of structure through:
  - the use of an overall architectural framework [Weston 92, Mertins 92, Kosanke 91];
  - the use of a layered decomposition (within the complete soft integrated vision machine) incorporating interfaces based on virtual machine abstractions [Seidewitz 86];
- The use of structured software where additional unforeseen requirements can be implemented within existing C code templates [Kernigan 88];
- An implementation based on the use of an integrating infrastructure where the infrastructure has underlying management facilities which can support change.

2.0 THE OPEN INTERACTION OF APPLICATION OBJECTS WITHIN A SOFT INTEGRATED VISION MACHINE

2.1 Introduction

Within open systems integration, the requirements for inter-application interaction can be met by the provision of a minimum range of services. These services should enable the generation of an association between two applications, the ability to request association status, and the ability to send data between associated applications. The Alien Device Driver must provide the functionality which enables the Remote Vision Service Provider to appear to Vision Client Applications as another open application. i.e. the Remote Vision Service Provider must comply with the above requirements for open interaction, and will thus offer vision services on the CIM-BIOSYS integrating infrastructure. To describe the functional requirements of the Vision Alien Device Driver, it is first necessary to detail the interaction aspects of the two processes which need to communicate - the Remote Vision Service Provider and the Open Vision Client Application. The following sub-section covers the interaction aspects of the Remote Vision Service Provider based on a client-server model using UNIX Sockets, while sub-section 2.3 describes the interaction aspects of open applications on CIM-BIOSYS.

2.2 The Remote Vision Service Provider, And Its Interaction With The Alien Device Driver.

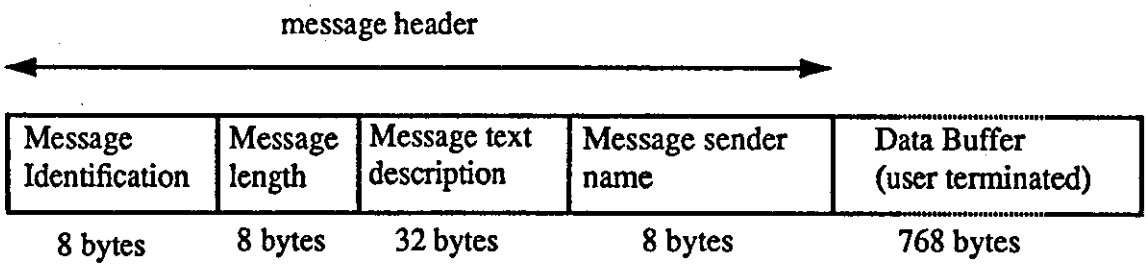
2.2.1 Introduction

As its name suggests, the Remote Vision Service Provider is a server which contains the communication interaction mechanisms so that clients can make use of the vision services offered by the vision application object. The socket based client/server IPC mechanisms [Sun 90a] exist within the "alien device server" and the "alien device client" identified within the remote vision server and the alien device driver shown in Figure 8, on page 193.

2.2.2 The Alien Device Client/Server Software Implementation Mechanisms.

The alien device server software within the remote system creates a server socket and "listens" for input of a "data packet" on the socket. The format of the data packet used is shown in Figure 1. The device server decodes the message header, and applies presentation layer services (OSI layer 6 [Pimentel 90], to convert the byte order of multi-byte variables) and checks the message received length. It returns an error message to the client if message errors are detected, or passes on the data buffer and message identification to the virtual vision machine.

FIGURE 1. Form of the Vision Client / Server data packet



The software to implement this functionality is written in Microsoft C. The particular C compiler, and the constraint of using C rather than C++, derives from the need to use proprietary PC-NFS software. This was required to provide the UNIX socket extensions to DOS on the PC machine.

The alien device client software resides within the alien device driver which runs on any general purpose Sun workstation on the network. When the alien device driver is invoked, its initialisation routine creates an IPC socket. It then builds and sends an "initialisation" data packet

containing a request for the status of the remote vision service provider. On receipt of a positive response from the server the alien device driver remains established to form its link between the vision server and open client applications on the CIM-BIOSYS infrastructure.

2.3 The Open Client Vision Application, And Its Interaction With The Vision Alien Device Driver Via CIM-BIOSYS.

2.3.1 Introduction

Both the Open Vision Client Application and the Vision Alien Device Driver require a general purpose processing resource and run on Sun workstations interacting via the CIM-BIOSYS integrating infrastructure. This section details the requirements for CIM-BIOSYS compliancy, i.e the use of CIM-BIOSYS services by open applications and their ability to field incoming CIM-BIOSYS messages.

FIGURE 2. Relationship between CIM-BIOSYS and user applications

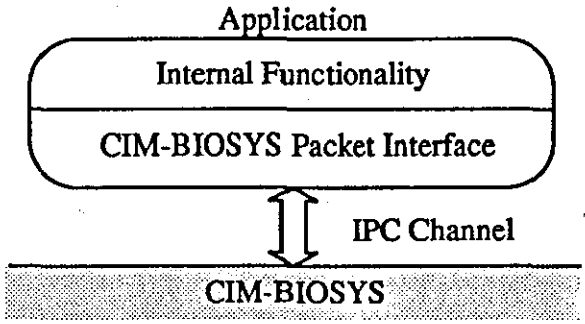


Figure taken from figure 3 reference Coutts 92

2.3.2 CIM-BIOSYS Interface Requirements

CIM-BIOSYS has been constructed to present a "clean" interface to applications, this imposes the minimum constraint on the architecture of the application i.e. a simple and consistent interface where no specific programming language is required by the user application. The service interface is implemented using an interprocess communications (IPC) channel currently based on connectionless sockets [Sun 90a]. The IPC transfers formatted data packets between the CIM-BIOSYS integrating infrastructure and open applications [Coutts 92] as shown in Figure 2. The contents of a data packet determine the nature of the message and how it should be treated. The ability to construct, send, receive and respond to CIM-BIOSYS data packets is

the only requirement imposed by CIM-BIOSYS on open applications. How this functionality is implemented is irrelevant to CIM-BIOSYS.

The approach described above provides maximum flexibility in the design and implementation of applications. However, more prescriptive facilities are available to support common application types [Coutts 92]. In the author's implementation a set of run-time 'C' library functions was used which typically handle IPC, packet decode/encode/buffering, and the support of Graphical User Interface development. The use of these libraries, is documented in the "Guide to writing CIM-BIOSYS System Applications" [Gilders 91].

Figure 3 shows the two areas of functionality required for CIM-BIOSYS compliant interaction:

- the use of CIM-BIOSYS interaction services i.e. calling functions for Establish, Terminate, Status and Send Data as described earlier, and;
- the provision of functionality to respond to incoming CIM-BIOSYS messages.

The Alien Vision Device Driver and the Open Vision Client Application interact by calling CIM-BIOSYS services and fielding CIM-BIOSYS messages. Both processes include an initialisation stage in which particular functions are registered with CIM-BIOSYS to handle the standard application interaction services.

Having described the interaction requirements of the Alien Device Driver at both its interface to CIM-BIOSYS and its interface to the Remote Vision Server, it is now appropriate to describe the overall structure of the Driver.

FIGURE 3. Typical Interactions of a CIM-BIOSYS application

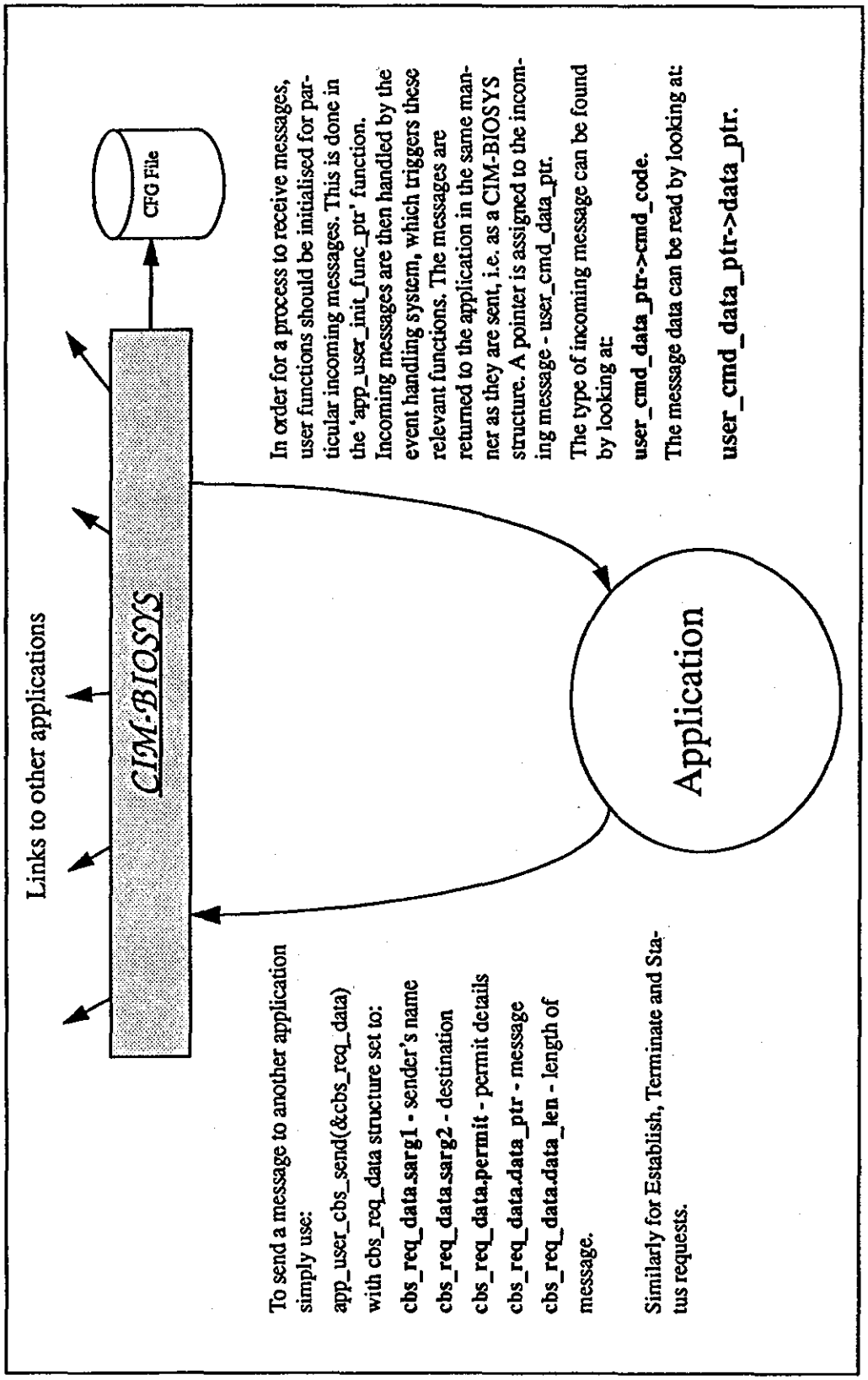


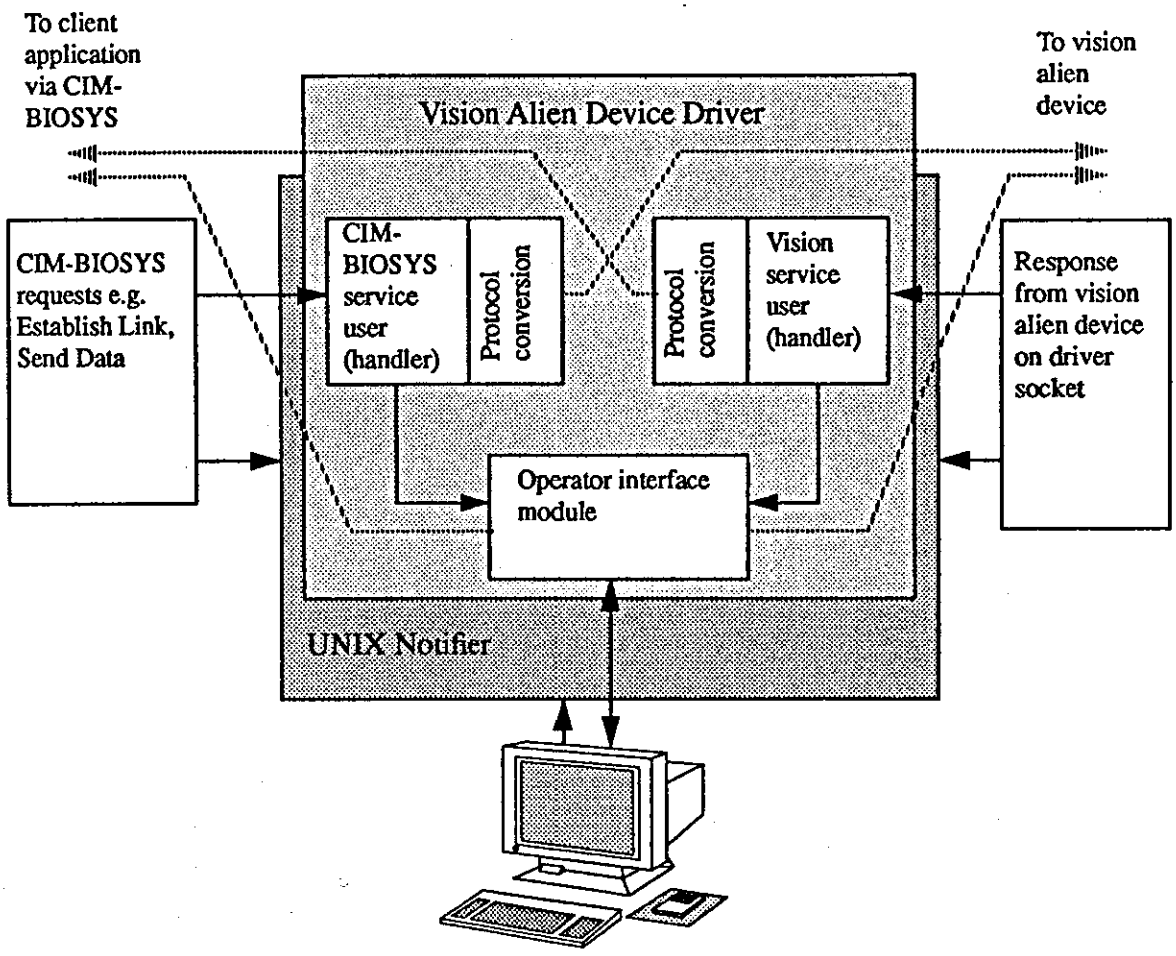
Figure taken from reference [Gilders 91]

2.4 The Alien Vision Device Driver

2.4.1 The Principal Event Driven Modules

The Alien Vision Device Driver has three principal functions which respond to external events, as shown in Figure 4.

FIGURE 4. Operation within the Vision Alien Device Driver



These can be considered as three discrete modules as follows:

- the interface to the remote vision service provider through the vision service user, which responds to the reply messages sent from the remote system (this module is equivalent to the alien device client service user layer in Figure 8);
- the interface to the CIM-BIOSYS integrating infrastructure through the CIM-BIOSYS service user, which responds to request messages sent via CIM-BIOSYS from open applications requesting services from the remote vision service provider (this module is equivalent to the CIM-BIOSYS service user layer in Figure 8);

- the Alien Device Driver operator interface which provides manual control and monitoring of the driver for debug purposes, and responds to requests via a mouse driven window interface on a Sun workstation.

Protocol conversion addresses the requirement to convert from CIM-BIOSYS compliant message packet format to the packet format used by the remote vision server, and vice versa. This functionality is driven by, and is essentially part of, the two service user modules identified above and shown in Figure 4.

The event driven operation of the three modules which implement this functionality is controlled by the "Notifier". The Notifier is a Sun tool [Sun 90b] which provides a mechanism for distributing events to a number of functions within a process. A function, and an object on which an event may occur (socket, keyboard, mouse etc.) are registered with the Notifier. When an event occurs on a registered object, the appropriate registered function is called.

Figure 4 shows the three event driven modules and their relationship with their external environment. The following sub-sections give details of the three modules.

2.4.2 The Vision Service Handler

This module performs much the same function as that detailed within section 4.2 describing the operation of the alien device server. The function of the module is to field data packets from the alien device server, apply error checking procedures, and to process the message. The principal process is to construct a CIM-BIOSYS data packet using the complete vision server data packet as data, and send it to the associated open client application via CIM-BIOSYS.

2.4.3 The CIM-BIOSYS Service Handler

This module is made up of a number of functions which are registered with CIM-BIOSYS and are called in response to incoming CIM-BIOSYS messages. The principal functionality is as follows:

- output message information to the driver operator interface, strips the vision client application message out of the CIM-BIOSYS data packet, and then send the vision client message to the remote vision server;

- terminate the association between the client application and the driver and then terminate the driver;
- return the status of the remote vision server;
- display the response messages from CIM-BIOSYS.

The establish link functionality is dealt with in the initialisation facilities detailed in Sub-Section 2.2.2.

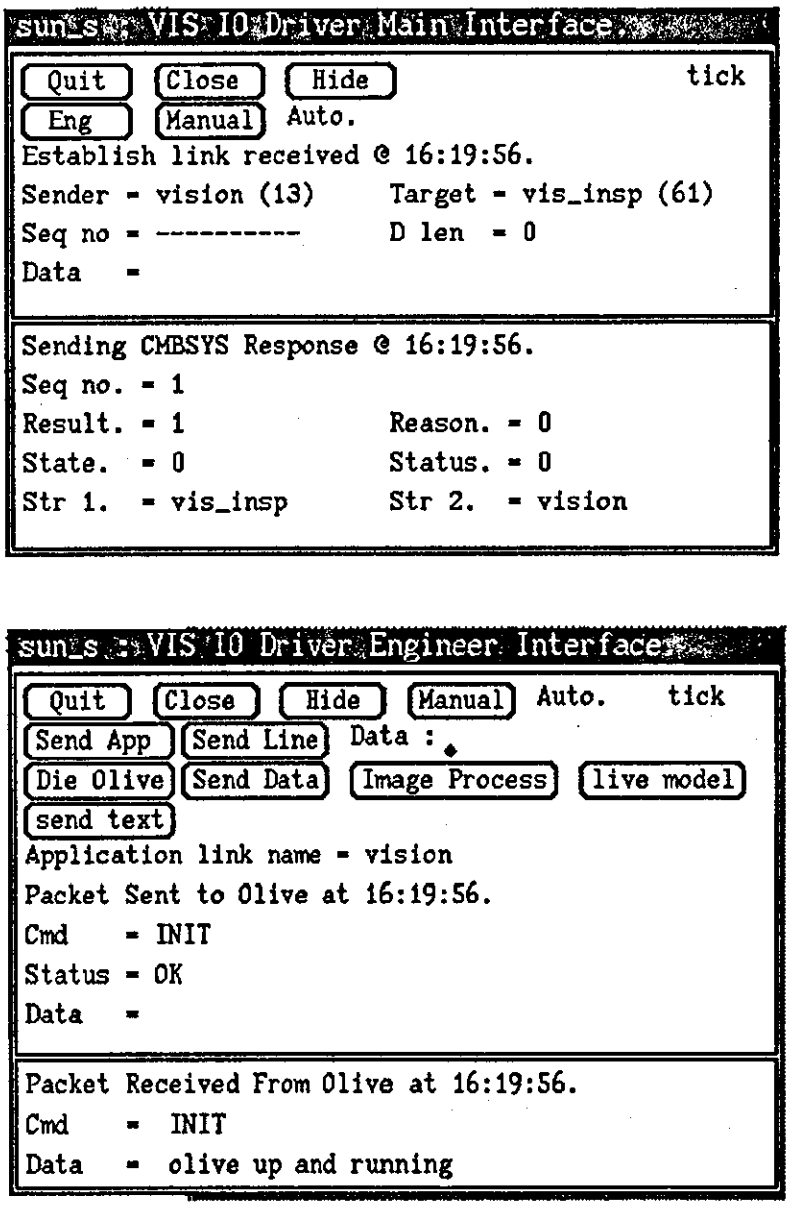
2.4.4 The Operator Interface

The Alien Vision Device Driver operator interface is implemented as a window based Graphical User Interface (GUI) comprising two windows. Bit map representations of these windows taken from a VDU are shown in Figure 5. The "Main Interface" is a general purpose window which displays CIM-BIOSYS interaction with the driver. It provides facilities for opening and closing the window, terminating the driver, or invoking the second window. Information pertaining to the establish link process is displayed in the upper half of the window. "Sender = vision" refers to the CIM-BIOSYS logical name for the Vision Client Application, while "Target = vis_insp" refers to the logical name for the Vision Alien Device Driver. The bottom half of the display provides information pertaining to the CIM-BIOSYS response.

The "Engineers Window" displays details of the messages passed to and from the remote vision server and a vision client application, it also enables manual operation of the driver. ("Olive" is the logical name for the PC Alien Device).

Figure 4 shows arrows from the two service handler modules to the operator interface. These represent the passing of message information for display in the windows. The figure also shows arrows from the operator interface module to both the Client Application via CIM-BIOSYS, and to the remote vision server. These arrows represent the manual operation of the driver where test and debug messages can be sent to the client application or the remote vision server. These facilities are invaluable for proving the correct operation of all the mechanisms involved in the transfer of messages over CIM-BIOSYS or over the remote link to the alien PC system.

FIGURE 5. GUI's for the Vision Alien Device Driver



3.0 THE INTEROPERATION OF THE VISION SERVER AND THE VISION CLIENT.

3.1 The Vision Server

3.1.1 A Virtual Vision Machine (VVM)

The VVM is that portion of the remote Vision Server that maps the services provided by the Vision Application Object on to the interaction mechanisms described in the previous section.

The nature of the VVM is then governed by the services provided by the Vision Application Object, and the interaction requirements of the remote device on which it is implemented.

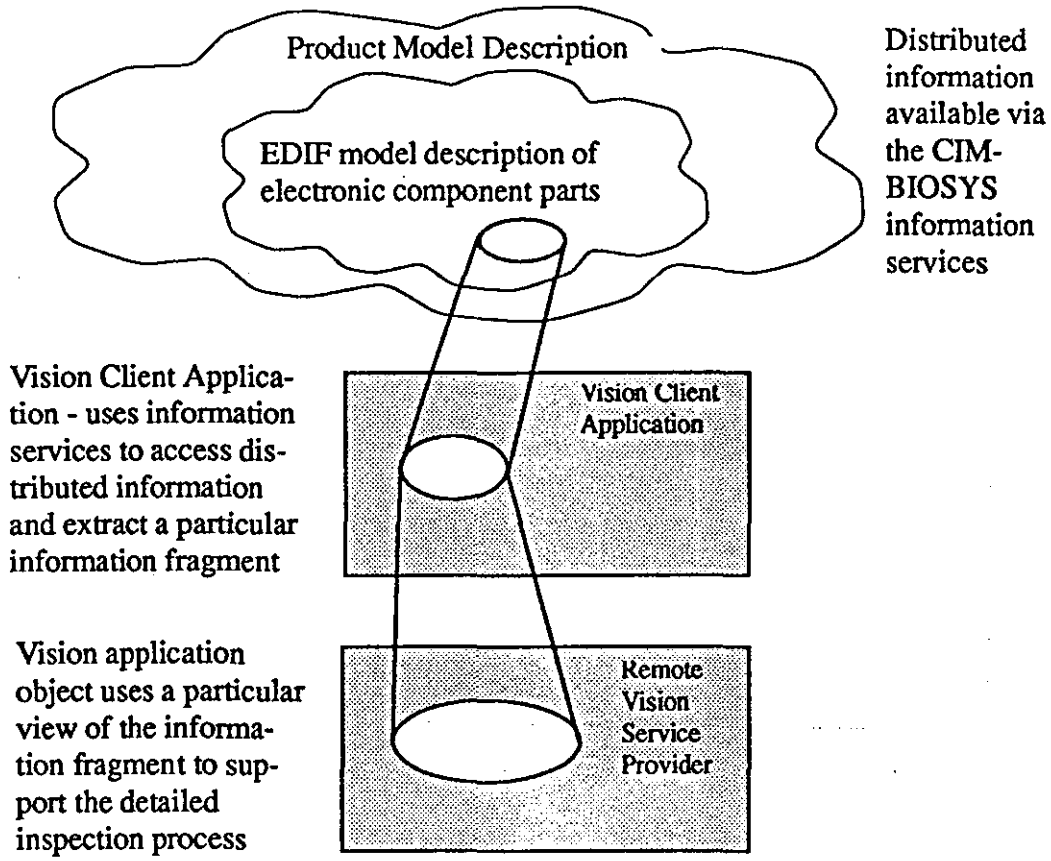
A vision machine can provide services which usually involve the processing of raw images, and can range from the provision of a set of application specific features of an object of interest, through classification decision information, to information relating to the understanding of the scene being viewed. The Vision Application Object described in chapter 7 extracts application specific features i.e. corner points of component package pins. The full specification of the Vision Application Object given in chapter 7 requires the following interoperation with a Client Application:

- Program Invocation;
- Information Upload, and;
- Information Download

The use of an EDIF file format for the upload of information provides an industry standard which has the potential to make a Virtual Vision Machine more generally applicable within the application domain addressed by this thesis. However, other implementations by the author extract more "vision generic" features, typically describing the package pins of a component in terms of an Arc Feature made up of a 'C' structure describing a Freeman chaincode [Pugh 83]. An International Standard specifying the form of vision features would provide support for a consistent form for the information to be uploaded from the remote vision server.

The form of downloaded information to drive vision machines is an application specific problem. The EDIF model provides an advantageous standard form for use in the application domain addressed by this thesis. Typically, an application specific view of design information describing the object under inspection will be required, and the form of this view will be specific to the requirements of the vision application object. Within the implementation described the specific view takes place within the application object, with the Client application having provided an information fragment or intermediate level view from the global information available on the Integration platform. Figure 6 describes this approach to information view provision.

FIGURE 6. two stage view provision



Other applications written by the author (not based on the EDIF model) have been of a more general nature. Typically the co-ordinates for window processing have been derived within the "information management" object running on the integrating infrastructure and have been down loaded to support vision processing. Standards defining the form of information required to support generic vision processes (e.g. window processing, thresholding) could provide support for the development of a generic form of download object for Virtual Vision Machines.

3.1.2 Implementation Mechanisms Within The Virtual Vision Machine

The Virtual Vision Machine (VVM) operates on the decomposed data packet which is passed to it by the Alien Device Server (see Figure 8). This information comprises the package header and data buffer (see Figure 1). The header contains the "msg_ident" field which controls how the VVM deals with the message. There are three important classes of message to implement the interoperation requirements identified. These interoperation objects are as follows.

3.1.2.1 The DATA Interoperation Object

DATA maps the message data within the data packet sent from the client into a data store (based on a 'C' structure) where the vision application object can access it for detailed feature extraction. The DATA object uses a variable name which is declared as a particular data structure type. In order to reconfigure the DATA object to operate on another data type the variable name is simply re-declared as the new data structure type. The only constraint on the nature of the data structure is the maximum length of the data string in the Vision Data Packet

3.1.2.2 The PROC Interoperation Object

PROC invokes the program named within the data packet. The VVM uses the "Spawn" facility within Microsoft C and DOS to handle program invocation. This allows the running of a separate "child" task and the return to the "Parent" VVM task on completion of the "child". This facility allows any program within the application object to be invoked and forms a clean separation between the VVM and the real Vision Application Object. This separation provides both flexibility in implementation and facilitates ease of change.

The flexibility offered is demonstrated within the author's implementation. Here the application object comprising the Object Oriented vision processing software is written in Zortech C++, while the vision server and VVM are written in Microsoft C, since this is a more generally available compiler as used by the PC-NFS communications software. The use of the spawn facility provides a solution that allows the real application object to be implemented in any programming language.

The facilities for ease of change are implicit within the implementation. The PROC message type contains the file name, and path to, the executable program to be spawned. The executable program has only to be present within the alien device. New programs can be added within the vision application object without the requirement for modification within the VVM.

3.1.2.3 The LIVE Interoperation Object

LIVE sends the live model information or application features generated by the vision application object to the Client Application. Facilities for reconfiguration are as in the DATA object, since the nature of the data handled by LIVE is determined by the 'C' data structure used. A

particular implementation which needed to send a large amount of data describing component legs as Arc Descriptors uses a series of messages (one for each leg), followed by a data structure describing the component leg count. Thus the requirements of a particular application can be fulfilled, overcoming the constraint of the message data packet size

3.1.2.4 Additional Interoperation Facilities

Extra facilities within the VVM are provided to support status requests, test/debug messages and a mechanism for releasing the server back to DOS (when running as a server the single tasking DOS machine is locked on to the server socket, listening for messages on the network).

3.2 The Vision Client or Vision Application Service Element (ASE)

Figure 8 shows the Vision Client Application has a layer dealing with the use of vision services. It is this area of functionality which engages in dialogue with the VVM. It must compose messages in accordance with the requirements of the VVM and be capable of interpreting and dealing with the form of reply received from the VVM. It receives data structures containing incoming replies from the function registered with CIM-BIOSYS to handle "send data" commands from the Vision Alien Device Driver (as described in section 4.3.1). It must also pass constructed messages to its CIM-BIOSYS service user layer which then encapsulates them within a CIM-BIOSYS data packet and sends them to the Vision Alien Device Driver.

This work recognises the need for a vision service user layer or vision ASE which can support the use of the VVM by open manufacturing applications. The author's work has implemented vision ASE functionality using a set of 'C' functions which manufacturing application builders could use to create vision client applications written in 'C'.

3.3 A Soft Integrated Manufacturing System Building Block For Machine Vision Inspection

All sections of the Remote Vision Service Provider and its Vision Alien Device Driver have now been introduced and described. The combination of these two components form a soft CIM building block as identified in the previous chapter. The building block can be "plugged into" a CIM-BIOSYS integrating infrastructure and used by any CIM-BIOSYS application.

The user or client application requires only the knowledge of the interoperation facilities offered, and packet structure expected by the VVM, as encapsulated within the vision ASE.

Appendix 7 describes a particular Vision Client Application which could complete the implementation of a Soft Integrated Vision Machine.

4.0 A MODIFICATION TO PROVIDE AN ADDITIONAL VISION SERVICE

4.1 Introduction

The change to a vision application object described in Chapter 8 provided an addition service to inspect the component manufacturer's identification code on an IC. A modification to provide a new vision service has two distinct parts. Modification is required to the vision processing related issues, and also to the integration related issues to make the new service available to client applications. The following subsection describes the modifications required to the integration elements within the complete Soft Integrated Machine Vision System. This work is measured relative to the author's early experiments in integrating distributed applications without the use of an integrating infrastructure.

4.2 Identifying The Elements Which Require Modification

Figure 7 shows the elements which make up a conventional integrated solution. The client server model is used to structure communication between a remote server application and any number of client applications. This arrangement is typical of the systems built by the author while developing the Soft Integrated Machine Vision System. This early work will be used to help quantify the benefits of the soft integrated system based on the CIM-BIOSYS integrating infrastructure.

Figure 8 (a repeat of Figure 9 in chapter 9) shows the elements of the Soft Integrated Machine Vision System. This figure helps to identify the extent of the required modification detailed in the following points:

- new facilities within the Vision Server, or Virtual Vision Machine in the Remote Vision Service Provider, will be required;

- new facilities within the Vision Service User, or Vision ASE in the Vision Client Application will be required, such that new open applications software can interoperate with the vision server.
- no modification is required within the Alien Device Server of the Remote Vision Service Provider;
- no modification is required within the Alien Vision Device Driver;
- no modification is required within the CIM-BIOSYS service user of the Vision Client Application

The points above illustrate how all the modification is required within those elements of the system which implement the interoperation and application functions. No modification is required to the facilities provided to handle application interaction. This highlights the importance of the decomposition between interaction and interoperation.

The facilities within the distributed system which support application interaction comprise routines for building message headers, data packets and complete message packets. They also include routines for sending and receiving data packets using CIM-BIOSYS. Figure 8 identifies these facilities as the Alien Vision Device Driver, the Alien Device Server within the Remote Vision Service Provider, and the CIM-BIOSYS Service User within the Vision Client Application. The Alien Vision Device Driver will treat any new messages in the same way that it deals with existing messages i.e. it transforms the message packet from Alien Vision Device form to CIM-BIOSYS compliant form, having no regard for the contents of the message packet.

The new service requires the types of generic interoperation facilities identified as necessary for distributed machine vision. In this case modification within the VVM in the Remote Vision Provider, and the vision ASE within the Vision Client Application can be minimal.

The following sections describe the modifications identified in the points above and demonstrate how the VVM/ASE combination implemented via the CIM-BIOSYS integrating infrastructure can support the modification process. The following two sections describe interoperation from ASE to VVM and from VVM to ASE respectively.

FIGURE 7. Elements within a conventional distributed system

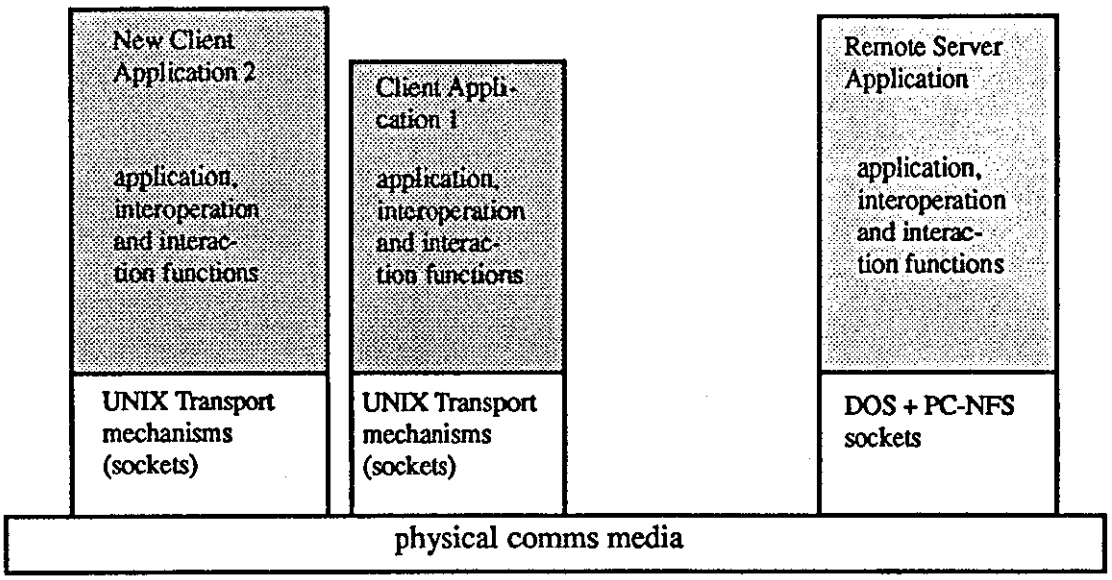
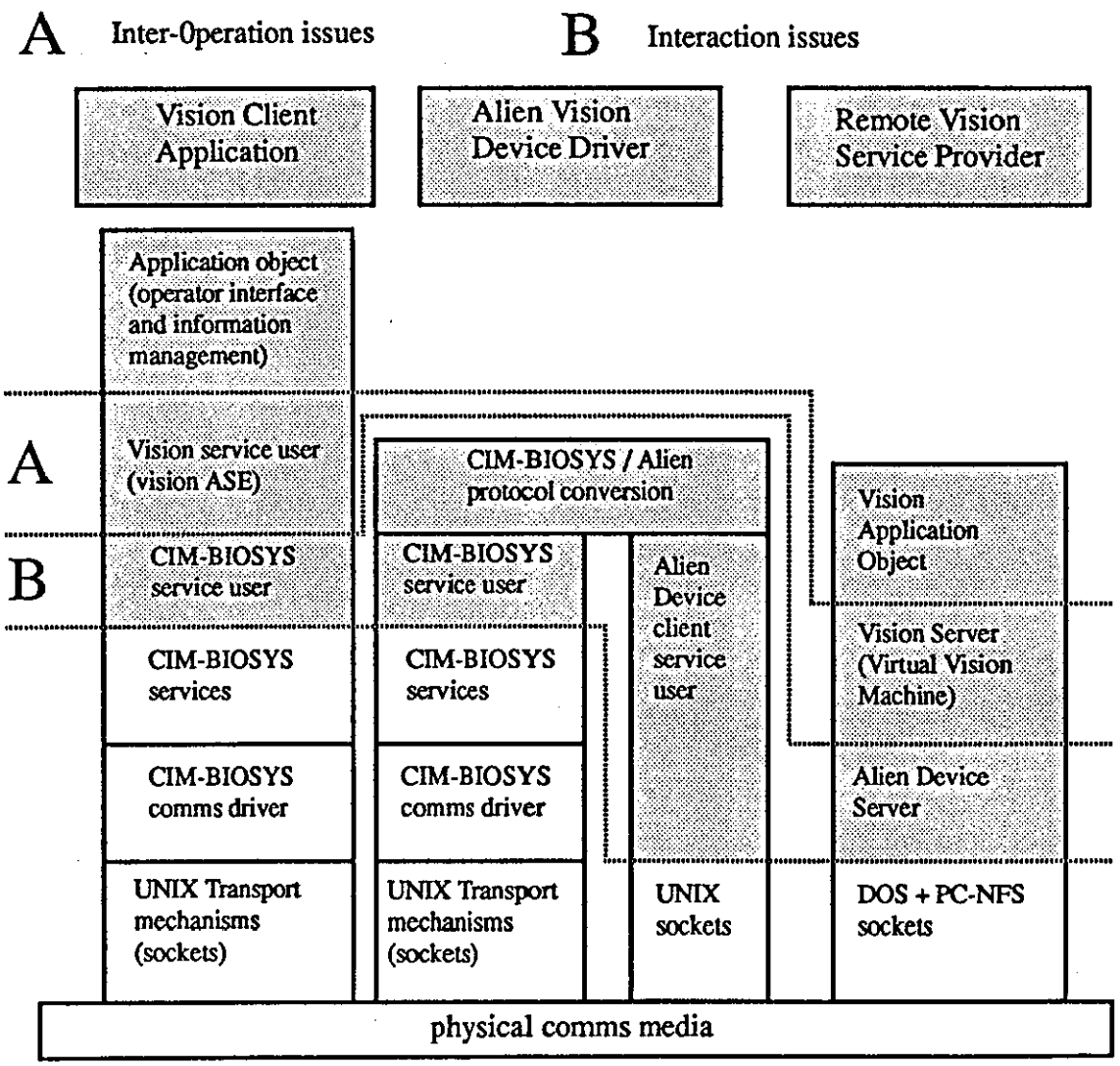


FIGURE 8. Elements within the Soft Integrated Vision System



4.3 Modification Of The Interoperation Requirements From The Vision ASE To The Virtual Vision Machine

The new facilities within the VVM will present a consistent interoperation interface, which includes the new service, to open applications running on the integrating infrastructure.

The main framework of the VVM code is a SWITCH structure [Kernigan 88] which provides alternative processing dependent on the message identification within the message header. Further structuring of the VVM code is incorporated within each CASE statement of the SWITCH structure. This extra structure provides further flexibility and support for change. Figure 9 shows a code fragment from the VVM which handles the LIVE message (a service to provide live model information to a request from a Vision Client Application). Within the LIVE - CASE statement a further SWITCH structure provides alternative processing dependent on options required within the interoperation of client applications and the VVM. In the author's modified implementation the flexibility provided by the option was implemented through the use of an additional C structure passed within the message Data buffer. Ideally a new message packet structure could provide specific facilities for flexibility within interoperation. Figure 10 provides a representation of the further decomposition within the facilities for interoperation, while Figure 11 proposes a new message packet structure which could provide improved support for the notion of decomposition between interaction and interoperation issues.

The structure within the Vision ASE comprises a function which is invoked by the client application together with a parameter which specifies the vision service required. The structure within the function is again a SWITCH statement which provides alternative processing dependent on the service required.

In order to handle the requirements of the new service all that is required is modification within the functions called by the CASE statements within the SWITCH structure, as these determine the structure and contents of the message data buffer.

FIGURE 9. Code fragment showing a CASE structure within the Virtual Vision Machine

```
case LIVE:
/*get switch info from msg datastructure*/

memcpy(&live_msg,(data_packin+sizeof(msgin_hdr)), sizeof(live_msg));
/*swap bytes sun to olive*/
live_msg.switch_name=ntohl(live_msg.switch_name);
/*respond to requirements of DATA msg*/
switch ((int)live_msg.switch_name)
{
case LEG_ARCS:
/*send live model message received */
/*special reply has to send complete model */
/*handled in file soicio.c*/
if (send_live_comp_model(sockin) < 0)
{
printf("error sending LIVE reply\n");
            close (sockin);
            exit(1);
      }
break;
case EDIF_LEG_CORNERS:
if (send_live_edifleg_model(sockin) < 0)
      {
                  printf("error sending LIVE reply\n");
                  close (sockin);
                  exit(1);
            }
break;
case STRING_ARCS:
if (send_string_arc_model(sockin) < 0)
            {
                  printf("error sending LIVE - STRING_ARCS reply\n");
                  close (sockin);
                  exit(1);
            }
break;
            }
break;
```

FIGURE 10. Structure within the facilities for interoperation

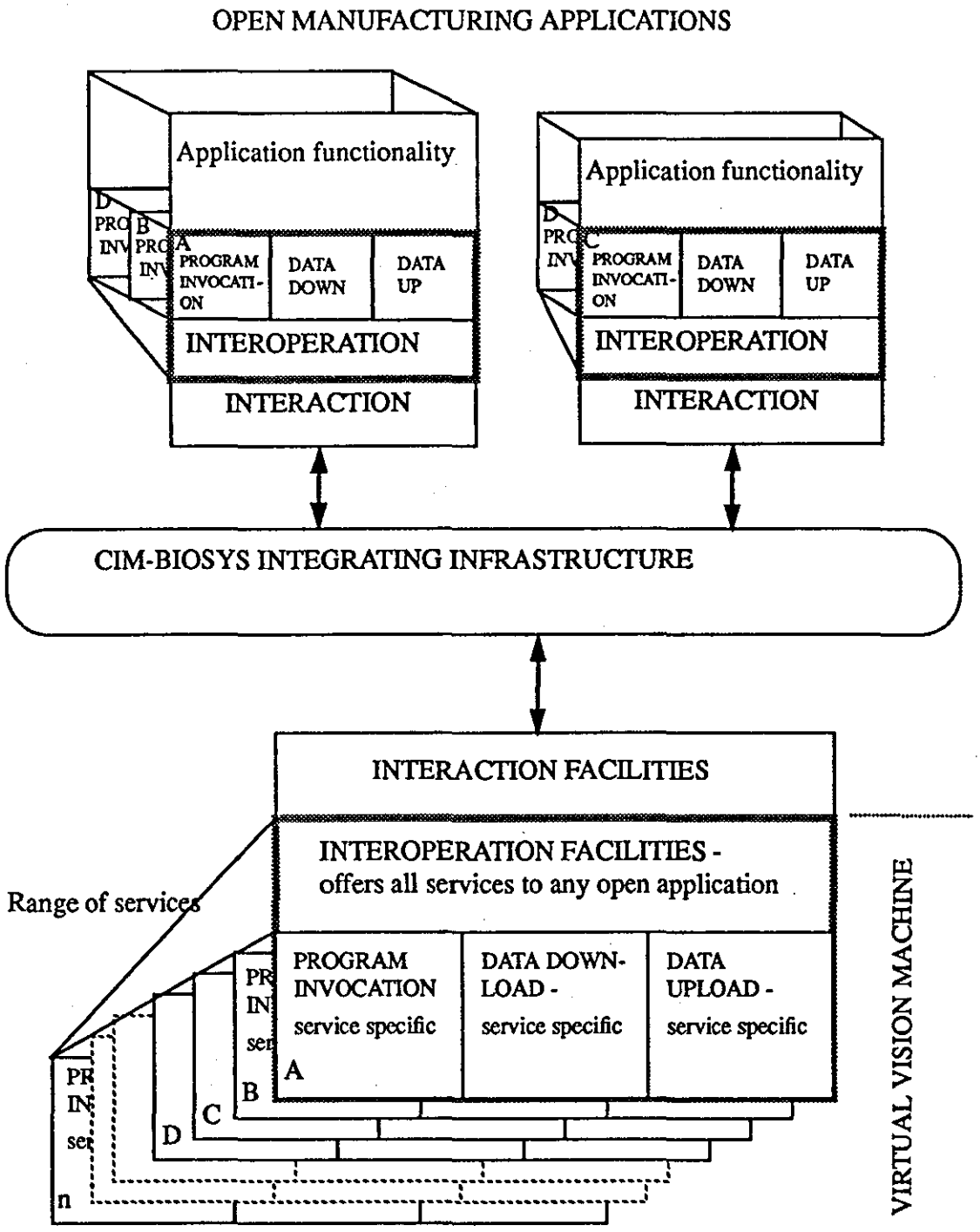
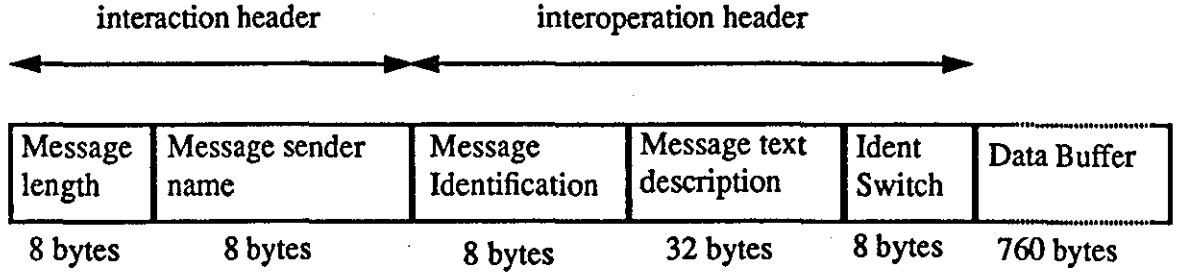


FIGURE 11. new packet structure supporting interaction and interoperation decomposition



4.4 Modification of the Interoperation requirements From The Virtual Vision Machine To The Vision ASE

The new facilities within the vision ASE will enable open applications to interoperate with the VVM and use the new service.

Vision services are always requested by the client and provided by the server. A specific service request decoded by the VVM will map onto a specific function which will complete the service. It is this function which controls the response when passing information back to the client.

In modifying the VVM to accommodate the information upload requirements of the new service an entirely new function is developed. The code fragment in Figure 9 shows the new function call (send_string_arc_model) within the modified LIVE CASE statement.

The Vision ASE uses a function which is registered with CIM-BIOSYS. The function is invoked when an incoming message arrives from CIM-BIOSYS for the client application. A parameter, passed with the function, is a pointer to the message packet constructed and sent by the VVM. Again, the main structure within the function comprises a SWITCH statement which provides alternative processing dependent on the Message Identification in the message header. This structure enables modification to handle the new service to be contained in a specific CASE statement of the SWITCH structure.

5.0 MODIFICATION WITHIN A CONVENTIONAL DISTRIBUTED SYSTEM

Figure 7 shows the elements of a conventional distributed system. In a system such as this it would be typical for all application functionality associated with communications issues to be contained within a single module. This module would typically be structured as a set of functions implementing the interaction and interoperation requirements of the system, as suggested by MacKinnon [Mackinnon 90]. The software to implement application interaction mechanisms within a distributed system are notoriously time consuming to develop and test.

A modification to implement a new client application which makes use of a new service provided by the server would require the implementation of the interaction mechanisms within

the new client application. This requires knowledge of the interaction mechanisms used within the server software. It is this re-implementation of interaction facilities that is overcome by the use of the Alien Vision Device Driver within the framework of the CIM-BIOSYS integrating infrastructure.

The interoperation functions within the client and server of a conventional system would typically be implemented through SWITCH structures as is the case with the VVM and Vision ASE of the soft integrated system. However, the identification of the generic functions within the VVM/ASE, together with the knowledge that the VVM/ASE is where new services are added are further elements of support that are missing in a conventional implementation.

6.0 RESULTS AND CONCLUSIONS

The changes made within the elements of the integration architecture in order to provide the new service to open applications on the integrating infrastructure took approximately 6 hours. The change made to the conventional IPC communications software took approximately 20 hours (primarily for re-implementing socket based IPC from scratch, in the new Client application).

6.1 Fulfilling The Needs Of Next Generation Machine Vision Systems

Change to the software which implements the integration aspects of the complete machine vision system clearly highlights the advantages of implementing distributed processing using an integrating infrastructure. In particular, it shows the advantage of implementing the application interaction mechanisms once only for any number of open client applications. The principal tool used within the integration study was the CIM-BIOSYS integrating infrastructure. An evaluation of the usefulness of this tool is presented in the following section.

The separation between interaction issues and issues of interoperation provides a modular decomposition where modification can be isolated within the mechanisms for interoperation. In the author's implementation these mechanisms comprise the VVM/ASE combination. Further structure within the VVM/ASE provides a modular template such that additional functionality can be easily added to provide new vision services.

The components described in PART C, which make up an integration methodology for vision machines, also provide support for design of new integrated systems. The overall architecture provides a reference framework, and the interaction facilities are domain generic and can be re-used. The VVM presents a consistent interoperation interface to new open client applications, while the ASE provides a set of ready built functions for inclusion within open applications to ease the use of the VVM services.

Heterogeneity is supported through the concept of virtual machines. Service specific software maps the VVM onto a particular Vision Application Object. Modification of this software enables a completely different Application Object, implemented using different vision hardware, a different implementation of the vision model, and a different application decomposition to replace an existing Vision Application Object.

6.2 The Usefulness Of The CIM-BIOSYS Integrating Infrastructure

The following points summarise the usefulness of the CIM-BIOSYS integrating infrastructure in underpinning the design, implementation and maintenance of the integration issues pertaining to distributed machine vision systems.

- **Soft Integrated Building Blocks of Manufacturing Systems:** CIM-BIOSYS provides an underlying framework which structures the creation of manufacturing applications (or application objects) so that they become building blocks of flexibly integrated manufacturing systems. These building blocks can be "plugged into" the CIM-BIOSYS infrastructure through "registration" with the CIM-BIOSYS configuration information. (An example of a CIM-BIOSYS Open Manufacturing Application could comprise an executable C program, but any executable program capable of handling UNIX inter process communication (IPC) is supported. It is this C program that is given a logical name and registered with CIM-BIOSYS).
- **Reconfiguration:** The reconfiguration management mechanisms within CIM-BIOSYS provide distributed vision applications, and other manufacturing building blocks which interoperate with them, with a means of supporting long term flexibility and change. Reconfiguration takes place through modification of the CIM-BIOSYS configuration

tables. This allows the association between logical applications and physical resources which make up the system to be modified without repercussion on the functionality of the distributed system.

- **Open vision services:** An Alien Vision Device Driver and its associated Remote Vision Service Provider provides a set of open vision services. These are application specific services offered at a level above that of the CIM-BIOSYS interaction services. Any open application registered with the infrastructure can access any of the vision services offered.
- **Heterogeneity:** The CIM-BIOSYS integration structure enables the inclusion of the Alien Vision Device Driver. Alien device drivers can enable flexible integration of legacy building blocks and map device specific communication technology onto the CIM-BIOSYS interaction services. This provides a consistent interaction interface through which open applications can interoperate.
- **Information access:** The soft integrated machine vision system described in PART C has demonstrated how the integrating infrastructure enables access to global information. This access promotes both the generation of information driven vision applications (which implies increased flexibility), and the generation of vision applications which make local information available for global use.
- **Overheads:** The use of CIM-BIOSYS implies an overhead in execution speed, size of the implemented system and time required to learn how to use the tool. Evaluation work which addresses the first two points (in relation to distributed manufacturing control systems) has been completed by the SI Group at LUT [MCS 93]. The runtime performance testing shows the overhead associated with the integrating infrastructure represents a small fraction of the overhead due to execution of the application systems themselves. Industrial trials of tools to support the design and build of CIM-BIOSYS compliant applications are imminent [MDC 93].

PART D

Chapter 11

CONCLUSIONS AND RECOMENDATIONS FOR FUTURE WORK

1.0 THE MAIN RESEARCH FINDINGS

The author's goal has been to define an overall architectural framework, supported by models, design methodologies and implementation mechanisms, which can significantly advance practice when flexibly integrating next generation machine vision systems within manufacturing environments. The work involved has covered a number of areas of study. In particular the author has:

- evaluated the contemporary role of machine vision systems within the electronics manufacturing industry, concluding that there is a need for both infrastructural facilities to support manufacturing systems integration, and a new generation of machine vision systems that can benefit from, and contribute to, an integrated manufacturing environment (chapter 2);
- reviewed past and present approaches to CIM system design and implementation, identifying the general acceptance, within the research community, of the need to build configurable soft integrated manufacturing systems through the use of an integrating infrastructure (chapter 3).
- reviewed contemporary machine vision implementation technology and its application in industry, concluding that there is a requirement to support vision application software design and implementation more systematically (chapter 4);
- assessed and published findings concerning opportunities for creating a new generation of integrated machine vision systems. (Appendix 8)

The thesis contributes to available knowledge concerning software design and implementation of machine vision systems, through the creation of a novel distributed machine vision inspection system based on the following proposals:

- that a layered architecture is required to structure processes contained within a vision application object. During implementation, mechanisms are required which support a clean interface between vision generic and vision application specific code;

- that an object oriented model, and associated design methodology, should be adopted, which covers image processing and feature extraction. This brings the benefits of object orientation and model driven design;
- that a design methodology is required to ease the identification of objects in the application domain;
- that a domain-specific architecture for structuring vision application software, should be adopted providing non rigorous structure which could lead to consistent design of application software;
- that a mapping is required between a more global CIM layered architecture and that of the machine vision layered architecture to support distributed machine vision systems implemented on an integrating infrastructure;
- that interaction issues and interoperation issues of objects within a distributed machine vision system be decomposed to form a Virtual Vision Machine and associated Application Support Element. This enhances the creation and maintenance of open distributed machine vision applications;
- that an alien device driver for handling the prevailing situation is required. This enables the open integration of vision applications running on processors which do not conform with the integrating infrastructure.

The thesis is based on, and has extended, the use of a number of recognised software engineering tools. The principal tools used are the object orientated paradigm, for structuring the vision application issues, and the use of an integrating infrastructure to underpin the design and implementation of the integration issues. The usefulness of these tools is evaluated in conclusions made in Chapters 8 and 10.

One of the original hypotheses of this work was that manufacturing applications such as machine vision should be designed to reflect the need for integration and the benefit that can be gained from available information. This work supports the conclusions that integration issues and issues of application functionality are entirely separate.

Study of PART B leads to the conclusion that manufacturing applications should be designed to be information driven and information providing, in the form of application objects. These applications should be built using a platform of techniques which offer support for rapid implementation, flexibility and change.

Study of PART C leads to the conclusion that integration methodologies in the domain studied are concerned with issues of application interaction. They are concerned less with the nature of the application i.e. assembly, inspection, test, but more with the nature of the hardware and software used to implement the application, i.e. PC, Robot, PLC, UNIX, C++ etc.

Issues of application interoperation form a new layer of functionality required for open distributed systems. Issues of interoperation are both integration and application related. Methods to support these issues have recently become a focus of global research.

2.0 FUTURE WORK

2.1 Application Interoperation

The interoperation of heterogeneous computer based processes within an integrated system has been identified as a major goal for the IT community. In the manufacturing domain the proposed solution to the interoperation problem has been through compliance with messaging standards such as MMS [MMS 90a, MMS 90b]. The work described in this thesis has identified the necessity to provide separation for interoperation issues from both the application issues, which drive interoperation, and the interaction issues which provide the media for interoperation. This separation, together with the increased demand for interoperating applications, implies a need for support facilities.

This thesis has demonstrated some simple mechanisms based on S/W templates which could support structured design, implementation and maintenance of interoperation. Future work could develop infrastructural software (an interoperating infrastructure) which would provide a framework and toolset to underpin message creation and interpretation. Such a system could provided the type of flexibility offered by integrating infrastructural systems in the domain of application interaction.

An interoperating infrastructure could contain knowledge of the interrelationships between applications and provide design, implementation, runtime management and reconfiguration facilities to support the lifecycle of the integrated system. Such an infrastructure could be used to implement existing standards such as MMS, but would support the creation of messaging systems appropriate to particular applications, and the use of existing de facto standards.

An interoperation infrastructure would offer similar advantages to those of the CIM-BIOSYS integrating infrastructure demonstrated in this thesis. Typically each application would send and receive messages from the interoperation infrastructure, not the related application. Also the framework would provide for the structured incorporation of "alien" messaging systems. The interoperation infrastructure would sit above the integration infrastructure making use of its integration services.

2.2 Linking Interactive Design To Documentation And Code Generation

Chapter 8 identified the benefits of using interactive image analysis systems to define an appropriate set of vision algorithms for solving a particular vision problem. The Booch based design methodology proposed in chapter 6 is of limited use for image processing design due to the requirement to experiment which is enabled through an interactive system. However, the use of the diagramming techniques proposed by the author have provided improved descriptive support which aids software modification. It follows that the linking of interactive image processing systems to the type of CASE tool technology that could produce Class and Object diagrams for documentation, together with executable code, represents a potentially useful field of study.

2.3 CASE Tools

Chapter 1 describes a scenario where open distributed systems can be designed and built by a group of co-operating software engineers who are situated in physically remote locations. This scenario could only be realised with the availability of fully automated CASE tool support for methods such as those described in this thesis. All the tools described within this work have been implemented by hand. Automated support for the Booch 91 methodology with the inclusion of the relationships devised by the author is an initial requirement. Further work is required to establish how to automate design and modification of machine vision systems based on the author's other structured mechanisms.

2.4 Domain Specific Models

The study and use of the "EDIF model for PCB" led to the identification of a four layered decomposition which was used to add consistent structure within the vision application software. The success of this technique, which contributed to the improved support for software modification detailed in chapter 8, implies that new areas of research concentrating on other

application domains could enhance the breadth of the architecture. This work could contribute to the derivation of a range of domain specific models for use within the applications layer.

2.5 Software Metrics

Chapter 8 has exposed the problems of measuring the degree of benefit offered by new proposals for supporting software engineering. Improved quantification of the author's proposals could be made through the use of controlled experiments with a group of software engineers who have a varying degree of expertise. Experiments based on the three variables identified by the author in Chapter 8 could provide a population of the frame work for quantification proposed by the author. This would provide a better feel for the usefulness of the proposals and the usefulness of structured support for machine vision software in general.

3.0 EXPLOITATION

Zwern's vision of the exploitation of automated inspection throughout a PCB fabrication shop [Zwern 90] can be extended to embrace automated inspection throughout an enterprise. Machine vision inspection could range from, the inspection of wafers within the semiconductor fabrication process [Jain 89] to the inspection of outgoing completed PCB's [ICL 93]. Information generated by these systems for local use is an increasingly valuable global enterprise resource. The author's proposed architecture for soft integrated machine vision can support these future systems, where multiple remote vision systems may share resources within a distributed vision application. Exploitation of this architecture is dependant upon a number of outstanding issues, including:

- manufacturing industry's recognition of the need to exploit available information;
- widespread acceptance of the need to build soft integrated systems based on an integrating infrastructure;
- the common availability of integration tools (an infrastructure and its support tools) which support the use of a wide range of processing hardware and software;
- the common availability and accepted use of CASE technology.

TERMINOLOGY

Acronyms and terms used in this thesis are defined as follows:

TERMINALOGY USED IN OBJECT ORIENTATION

base class - a class with general characteristics designed to be a parent class to several sub-classes;

class - a definition describing an abstract data type and its behaviour which in turn defines the interface to all the operations that can be performed on or with the underlying type;

constructor - a method invoked during the creation of an object, which populates its data type to some initial state;

friend - selected methods or an entire class can be declared to be a friend of another class. This relationship will allow them access to the private data of the friend class;

message - a call from one object to invoke a method of another object;

methods - the operations defining the behaviour of a class;

object - a particular instance of a class, which encapsulates state. An object has a populated class data type;

overload - method overloading is defined as polymorphism, (this is the ability to define the same method name for parent classes and subclasses, where the functionality of these methods is different);

subclass - a class that characterises the behaviour of a set of objects which inherit characteristics of the parent class but acquire specialised characteristics particular to the subclass;

GENERAL TERMINOLOGY

application interaction - the passing of data packets (or messages) between two applications;

application interoperation - Meaningful dialogue between two or more applications, (enabling the creation of an integrated aggregation of inter-working applications which combine to produce some greater functionality);

browsing facilities - facilities to enable particular elements of text to be found within the complex file structures which are inevitable when developing complex software systems;

CAD - acronym for Computer Aided Design, the use of computers for the creation and modification of engineering designs;

CAM - acronym for Computer Aided Manufacture, the use of computers for the management or control of manufacturing facilities;

CASE - Computer-Aided Software Engineering (tools) - used to speed up and formalise the process of system analysis and software design. Such systems use a variety of representations such as data-flow diagrams, entity-relationship diagrams and, in some cases, generate program code;

client / server model - A model for communication transport mechanisms where a server entity is permanently established on a network. Client entities form associations with the server when they require the services on offer;

data packet - A data entity made up of a number of fields which is passed from one application to another during application interaction;

design or library sources - will refer to information sources within an enterprise information system, typically a library of component part descriptions, or PCB designs;

elementary features - will comprise the perimeter, area and centroid of an unclassified component being inspected by the demonstration system;

ESPRIT - acronym for the European Strategic Program for Research and Development in Information Technology;

flexibility - The ability of a manufacturing application to handle short term reconfiguration within a predictable range of variation;

global information - will mean information within the enterprise information system which resides within information stores (databases or files) which can be accessed by open applications via the integrating infrastructure;

golden board - the name given to a printed circuit board which is known to be in specification and can thus be used as a reference;

hybrid PCB - A PCB which includes both SMT and through hole technology components;

integration - The aggregation of resources and applications into a synergistic whole;

integration toolset - A set of complimentary software application programmes to assist in or enable some aspect of the development or management of CIM systems;

IPC - interprocess communication;

islands of automation - stand alone element of automation within a manufacturing enterprise, typically covering a single NC machine, a group of machines within a work cell, a CAD system;

live model: will refer to a description, in model form, of a particular live PCB;

live PCB - will refer to a particular printed circuit board which is under fabrication within the manufacturing system;

make facilities - facilities which support the controlled compilation and linking of programs, essential for all but the simplest program development;

manufacturing application - some combination of processing hardware, software, interface components, machine and human mechanisms which make up a manufacturing process or

function, typical examples include robots, human inspection stations, cell controllers, and process planning systems;

MAP - acronym for Manufacturing Automation Protocol, specification for communication within the manufacturing environment, based on the International Standards Organisation (ISO) 7 layer Open Systems Interconnection (OSI) reference architecture, initiated by General Motors;

mechanisms - will refer to software implementation mechanisms;

methodologies - Formal methods of software analysis and design;

model - Within this thesis the term "model" is primarily used to describe some system of entities using a formal modelling methodology which usually comprises a set of diagramming tools, such as the "EDIF model for PCB" [EDIF 90]. The term can also be used to describe a less formal description such as a simple software template to ease code generation;

MRP II - acronym for material requirements planning, planning the resources of a manufacturing company, using simulation for optimization;

ODP - Open Distributed Processing;

OSI - Open Systems Interconnection;

open systems - Systems comprising a set of interacting open applications which communicate using mechanisms that adhere to relevant international agreed standards. These standards being non-proprietary and vendor independent. Applications can be removed, replaced or added to the system in a managed way. Usable information can be exchanged between applications;

onsertion - assembly of surface mount components on PCB's;

PC - personal computer;

PCB assembly - All processes involved in the mounting of electronic components on PCB's to realise populated and finished PCB's. This includes solderpaste and/or glue application component onsertion and/or insertion and soldering, together with inspection operations throughout;

PCB fabrication - All processes involved in the manufacture of multilayer printed circuit boards (PCBs). This will include the generation and use of phototools, the chemical processing in the creation of separate PCB layers, the bonding of the finished board, the drilling of the layers and finished board and inspection and test throughout the process;

phototools - The artwork transparencies describing the track and pad features of a layer within a PCB. These are used within the PCB fabrication process;

pixel - An element of a matrix which makes up a pictorial representation of a scene. Used to describe a single point within a graphical display or a single point within an image captured in a frame store;

product realisation - This term is used to imply a broader coverage than just shop floor manufacture. For example product design and introduction, finance and accounting, logistics, and manufacturing planning and control are also included in the definition;

reference architecture - An overall framework that may embrace models, methodologies and mechanisms which can support the lifecycle of their target domain;

SMT - Surface mount technology. PCB manufacturing technology where electronic components are connected to PCBs using solder to attach them to PCB component pads on the surface of the PCB;

stereo vision - binocular imaging through the use of two cameras to generate different images of the same scene. Points in the two scenes are matched and depth information is computed to construct a 3D representation of the scene;

TCP-IP- This is an 'internet' protocol which can run over the top of a number of different types of local area network (e.g. Ethernet and SNA) making their distinctive characteristics transparent to the user;

TOP - acronym for Technical Office Protocol, specification for communication within the office environment, based on the ISO/OSI reference architecture;

touch up - a PCB assembly process which follows final inspection (or is incorporated within it). The process involves minor hand finishing of the board, typically to make good poorly soldered joints;

through hole plating - is the process of etching tin to the inner wall of holes in PCB's, so forming a conducting link between layers within a multi layer PCB;

through hole technology - Conventional PCB manufacturing technology where electronic components have leads which pass through holes in the PCB before solder is used to make electrical connection between component lead and PCB pad;

wash off - a PCB assembly process which usually follows flow solder and precedes final inspection prior to test. The process washes off any contamination from the PCB.

REFERENCES

Abbott 83: Abbott R., "Program design by informal English descriptions",Communications of the ACM,Vol26(11), Nov. 83.

Aguiar 92: Aguiar M.W.C., 1992, "First exercise on the application of the CIM-OSA modelling methodology" SI Group internal document, available from LUT.

Allderdice 85:Allderdice H.B., King R.I., "The design of a computer integrated electronics manufacturing system", Computer Aided Engineering Journal, April 1985.

Analogic: Analogic Sun-3/E Compatible image processor catalogue, available through Analogic, CDA division, 8 Centennial drive, Centennial Industrial Park, Peabody, MA 01961, USA.

Annaratone 86: Annaratone M.,Arnould T., Gross T. et.al., 'WARP Architecture and implementation", Proc. 13th Annual Symposium on Computer Architecture, p346-356, June 1986.

Azar 88: Azar I. Weston R.H., "A vision reference model for systems integration", I.J. CIM, Vol. 4, 1988.

Azar 89: Azar I, "A vision architecture for integrated manufacturing systems", Ph.D. Thesis from Loughborough University of Technology, 1989.

Ballard 82: Ballard D.H. and Brown C.M., "Computer Vision", Prentice Hall,1982, ISBN 0-13-165316-4.

Batchelor 85: Batchelor, B.G., Hill, D.A., Hodgson, D.C.,1985, Automated Visual Inspection, IFS (Publications) LTD, UK. ISBN 0-444-87577-5.

Bell 92: Bell Prof. R., Popplewell K., "Report of Japanese Study Visit", funded by ACME directorate SERC, Contract GR/F 96549, 1992. Available from the Dept. of Manf. Eng. Loughborough University of Technology.

Bessant 88: Bessant J., "The ups and downs of implementing CIM", "CIM - Mechanical Aspects", Pergamon Infotech LTD, ISBN 0 08 034116 0

Bilow 93: Bilow S.C., "Software entropy and the need for object oriented software metrics", Journal of Object oriented Programming, 1993, Vol5, No 8, P 6.

Bishop 89: Bishop T. Schofield N., "Unlocking the potential of CIM" PA Consulting Group, A management guide to creating business advantage. 1989. Available from PA Consulting, Cambridge, UK.

Black 85: Black A., "Supporting Distributed Systems", ACM Operating Systems Review, 19(5),181-193 (December 85).

Black 88: Black R.J., "Versatility and vision: The key to accurate placement of surface mount packages", Electon. Product. Pruftech., no 8,1988.

Blakeslee 89: Blakeslee A.B., "Verification: the neglected portion of AOI", PC Fabrication journal, December 1989, 24-30.

Bracker 89: Bracker W.E., Harris C., "Automatic optical inspection", Printed Circuit Assembly Journal, December 1989, 6-11.

Brady 92: Brady M. and Wang H., "Vision for mobile robots", Phil. Trans. Royal Society. London. B(1992) 337,341-350.

Brenner 87: Brenner J.B., "Open Distributed Processing", ICL Marketing and technical survey, ICL Technical Journal November 87.

BICS 87: "The British Industrial Computing Survey", Industrial Computing, EMAP Business and Computer Publications Ltd., Oct. 1987.

Boehm 78: Boehm B. et.al., "Characteristics of S/W quality", North-Holland, New York.

Booch 91: Booch G., "Object Oriented Design with applications", Benjamin Cummings Publishing Co., 1991, ISBN 0-8053-0091-0.

Bunce 88: Bunce P., Davenport F., "CIM - the next 5 years", "CIM - Mechanical Aspects", Pergamon Infotech LTD, ISBN 0 08 034116 0.

Camp 76: Camp J.W. and Jenson E.P., "Cost of Modularity", Proc. of the symposium on computer software Engineering, 1976.

Capson 90: Capson W.D., Tsang R.M.C., "Automatic Visual Measurement of Surface-Mount Device Placemen",1990, IEEE Transactions on Robotics and Automation. Vol. 6, No 1, February 1990.

Carlson 92: Carlson I.C. and Haaks D., "Concept and implementation of an object oriented framework for image processing", Philips Journal of Research, Vol. 46, No 6, 1992.

Carter: Carter M.K., et.al. The design and implementation of a portable image processing algorithm library (IPAL) in fortran and C, Report from Alvey project number MMI/127.

Cecchini 90: Cecchini R., Bimbo A.D., Nesi P., "Object classification and image interpretation with an object oriented system", University of Florence, Italy, 1990.

Chen 90: Chen S., "Automatic solder inspection and process control based on established visual criteria", Printed Circuit Assembly Journal, March 1990, 36-42.

Cheng 90: Cheng H., Rattner L., "Information Modelling for Computerised Manufacturing", IEEE Transactions on Systems, Man and Cybernetics, Vol. 20, no 4, July/August 1990, 758-776

CIM-OSA: ESPRIT project nr 688, CIM-OSA Reference architecture specification.

Clements 91: Clements P., "Information Systems Modelling and implementation in an industrial environment", Proc of Autofact 91, Chicago, USA, Nov. 1991.

Clements 92: Clements P., "The application of EXPRESS modelling and tools within an integration platform", Proc of EXPRESS users group 92, Dallas, USA, Oct. 1992.

Coad 90: Coad P and Yourdon E., "Object Oriented Analysis", Yourdon Press, Prentice Hall, 1990, ISBN 0-13-629122-8.

Coad 91: Coad P., "New advances in object oriented analysis", Journal of Object-Oriented programming, Jan. 1991, Vol. 3, No 5.

Cohen 91: Cohen R., "AOI: Quality watchdog and process overseer", Printed Circuit Fabrication Journal, August 1991.

Coutts 92: Coutts I, Weston R.H., Murgatroyd I.S., Gascoigne J.D., "Open Applications within Soft Integrated Manufacturing Systems", Proc. Int. Conf. on Manf. Automation, Hong Kong 1992.

Cook 91: Cook S., "Object Orientation", Conf. on Exploiting Object Oriented technologies, Brunel, UK, Feb. 1991.

Cox 87: Cox B.J.,1987, "Object Oriented Programming", Addison Wesley, ISBN 0-201-10393-1.

Danon 91: Danon O. and Kochba J., "Automatic optical inspection of drilled holes", Printed Circuit Fabrication Journal, February 1991, 76-78.

Davis 91: Davis S. and Davidson B., "2020 Vision Transform your business today to succeed in tomorrow's economy", Business Books Ltd., ISBN 0-7126-5089-X.

Dear 88: Dear A., "Working towards Just-In-Time", June 1988, Chapman and Hall, ISBN 1-85091-650-0.

DEC 92: Digital Equipment Corporation, "Network Application Support(NAS): an open system for application integration", support documentation, 1992.

Dean 91: Dean P., Mayhew J.E.W., "cascade control in a simulated robot camera-head system - neural net architecture for efficient learning of inverse kinematics', Biological Cybernetics 1991, Vol. 66 No 1 pp 27 - 36.

Deming 86: Deming W.E., "Out of the Crisis", MIT Centre for advanced engineering study, Cambridge, Mass., USA, 1986.

Dijkstra 68: Dijkstra E.W., "The structure of 'THE' Multiprogramming System", Communications of the ACM, May 1968.

Doyle 90: Doyle K., "The Next Generation of AOI", PC Fabrication Journal, September 1990, 89-93.

Doyle 91: Doyle K., "AOI: the way forward", Electronic Production Journal, April 1991, 17-21.

Duff 86: Duff j.b., Fountain T.J., "Cellular logic Image Processing", Academic Press, New York, 1986.

Durrant-Whyte 92: Leonard J.J., Durrant-Whyte H.F., Cox, I.J., "dynamic map building for an autonomous mobile robot", Int. Journal of Robotics Research, 1992, Vol. 11, No 4,

EDIF 90: "Conceptual Model of a PCB", EDIF PCB Technical Sub-committee, version 9. 1990, available through the Electronic Industries Association.

EDIF 91: Information Model of a PCB (EXPRESS), produced by the EDIF PCB Technical Sub-Committee, Version 9, 1991, available through the Electronic Industries Association.

Edwards 90: Edwards J., "Machine vision and its integration with CIM systems in the electronics manufacturing industry", Computer Aided Engineering Journal, Feb. 1990,12-18.

Edwards 93: Edwards J, Clements P., Murgatroyde S., "Machine vision integration and information support - methods models and tools". Int.J.Computer Integrated Manufacturing, 1993, Vol.6, No.5, 323-334.

Eliott 93: Eliott B. "Keynote Address: Computer Integrated Manufacturing in the Plastics Processing Industry", Proc. of ICIMS 93 Beijing, 1993.

Eustace 87: Eustace P., "Counting the cost of CIM", The Engineer, July 1987,

Evans 88: Evans D. and Perez E., "Unisys on CIM", CIM - Mechanical Aspects, Pergamon Infotech Ltd., 1988, ISBN 0-08-034116 0.

Fairhurst 88: Fairhurst M.C., "Computer vision for robotic systems", Prentice Hall 1988, ISBN 0-13-166927-3.

Fenton 91: Fenton N.E., "Software Metrics: A rigorous approach", Chapman Hall, 1991, ISBN 0 412 40440 0

Flat 91: Flat M.O. and Holden H., "What will a world class PWB Shop look like in the year 2000", Printed Circuit Fabrication Journal, September 1991, 48-50.

Foster 90a: "C-Images for the Matrox MVP-AT - user manual", Foster Findley Associates, Newcastle upon Tyne, UK, 1990.

Foster 90b: "PC-IMAGE for the Matrox MVP-AT - user manual", Foster Findley Associates, Newcastle upon Tyne, UK, 1990.

Freeman 74: Freeman H.,"computer processing of line drawing images", computer surveys 6,1,march 1974,57-98.

Fu 87: Fu K.S.,Gonzalez R.C, Lee C.S.G., "Robotics: Control, Sensing, Vision, and Intelligence", McGraw-Hill, 1987, ISBN 0-07-100421-1.

Fuji 88: Information on the Fuji IP-I 4000 Fuji Multi Chip Placer, information provided Astro Technology Ltd., Fareham, Hampshire 1988.

Fuji 89: Fuji Machine MFG. CO. Ltd., Multi Chip Placement general catalogue, available from Astro Technology Ltd. Fareham, Hampshire U.K.

Gascoigne 92: Gascoigne J.D.,1992, "CIM-BIOSYS, its purpose and functional overview", SI Group Internal Document, available from Manufacturing Eng. Dept., Loughborough University of Technology.

Gilders 92: Gilders P.,Jan 1992, "Application of CIM-BIOSYS to the NIST MSI project", SI Group Internal document available from LUT Manf. Eng.

Gilders 91: Gilders, P., "Guide to writing CIM-BIOSYS System Applications", SI Group Internal Document, available from Manufacturing Eng. Dept., Loughborough University of Technology.

Gilutz 91: Gilutz H., "AOI for high density circuits", Electronic Production Journal, July/August 1991, 31-33.

Gilb 87: Gilb T., "Software Metrics", 1987, Winthrop.

Gillies 92: Gillies A.C., "Software Quality: theory and management", Chapman Hall Computing, 1992, ISBN 0 412 45130.

Goetsch 90: Goetsch D.L., "Advanced Manufacturing Technology", Feb. 1990, Chapman and Hall, ISBN 0-82733-786-8.

Gonzalez 77: Gonzalez R.C. and Wintz P., "Digital Image Processing", Addison Wesley, 1977.

Harrington 79: Harrington J., "Computer-integrated manufacture", 1979, Robert E Krieger Publishing Co., Huntington NY USA

Hodgson 90: Hodgson, R., November 1990, Object Oriented Approaches to Reuse, UNICOM "Software reuse and reverse engineering in practise" Seminar, London 3-4 December 1990.

Hollingum 86: Hollingum, J., 1986, The MAP Report, (Bedford, IFS Publications Ltd.)

Horn 79: Horn B.K.P., "A.I and the science of image understanding, in: computer vision and sensor-based Robots, Ed. G.dodd and L.Rossel, Plenum Press, New York

Hutton 91: Hutton W., Economics editor of The Guardian newspaper London UK., Lead articles on the Economics page November/December 1991.,

IBM 87: "Computer integrated manufacturing: the IBM experience", Findley Publications for IBM UK Ltd., 1987.

ICL 89: Conversation between the author and Mr. Chris Sherrat of ICL Ltd. Kidsgrove UK.,during preparation of reference [Edwards 90], April 1989.

ICL 93: Conversation between the author and Mr. Brian Samual of ICL Kidsgrove UK, February 1993.

Imaging: Imaging technology general catalogue, available from Imaging Technology Incorporated, 600 West Cummings Park/Woburn, MA 01801, USA.

IP 93: Machine vision survey, Image Processing journal, Spring 1993, p16-17.

ISO 78: International Standards Organisation "7 layer Reference Architecture for Open Systems Integration", available through ITU, Place des Nations, 1211, Geneva, Switzerland.

Jain 89: Jain R. et. al., "Machine vision for semiconductor wafer inspection", Machine Vision for inspection and measurement, Academic Press, Inc. 1989, ISBN 0-12-266719-0.

Jorysz 90a: Jorysz h. r. and Vernadat F.B., 1990, "CIM-OSA Part1: total enterprise modelling and functional view", Int. J. CIM.,1990, vol. 3, nos 3,4, 144-156.

Jorysz 90b: Jorysz h. r. and Vernadat F.B., 1990, "CIM-OSA Part 2: information view", Int. J. CIM.,1990, vol. 3, nos 3,4, 157-167.

Katz 91: Katz T., "Engineering design manufacture and test", "Artificial intelligence in Engineering", John Wiley and Sons 1991, ISBN 0-471-92603-5.

Kernigan 88: Kernigan B.W. and Ritchie D.M., "The C Programming Language", Prentice Hall, 1988, ISBN 0-13-110362-8.

Klittich 90: Klittich M., 1990, "CIM-OSA integrating infrastructure-the operational basis for integrated manufacturing systems", Int J. CIM., 1990,vol 3, nos 3,4, pp168-180.

Kosanke 91: Kosanke K.,1991, "The European approach for an Open Architecture for CIM (CIM-OSA)-ESPRIT project 5288 AMICE", computing and control engineering journal, May 1991.

LaRonde 90: LaRonde W.R. and Pugh J.R. "Inside Smalltalk", Prentice Hall, 1990, ISBN -Vol1, 0-13-468430-3, vol2, 0-13-467309-3.

Lipietz 93: Lipietz A, "Towards a new economic order; Postfordism, Ecology and Democracy", Polity Press.

Lloyd 89: Trackscan 3000 catalogue, Lloyd Doyle Ltd., Walton on Thames, Surrey UK.

Lloyd 93a: Trackscan 3000 Description and specification sheet, Lloyd Doyle Limited, Russel House, Mosely Road, Walton on Thames, Surrey, UK.

Lloyd 93b: Conversation between the author and Dr. K. Doyle (Technical Director) and Mr. R. Lloyd (Managing Director), Lloyd Doyle Limited, Russel House, Mosely Road, Walton on Thames, Surrey, UK, March 1993.

Lochner 90: Lochner R.H. and Matar J.E., "Designing for Quality", Sept. 1990, Chapman and Hall, ISBN 0-412-40020-0.

LSI 93: Loughborough Sound Images, Digital Signal Processing and DA solutions for VME bus catalogue, available from Loughborough Sound Images, The technology centre, Epinal Way, Loughborough, UK.

MacKinnon 90: MacKinnon D. McCrum W. Sheppard D.,1990, "An Introduction to Open Systems Interconnection", Computer Science Press,W.H. Freeman and Co. New York. ISBN 0-7167-8180-8.

Magee 89: Magee J., Kramer J., Sloman M., "Constructing Distributed Systems in Conic", IEEE Transactions on Software Engineering, Vol 15, No 6, June 1989, 663-675.

Mansel 88: Mansel A., "CIM - a business impact", CIM-mechanical aspects, Pergamon Infotech Ltd,1988, ISBN 0-08-034166-0.

MAP 88: MAP and TOP Version 3.0 Specification. Obtainable from the society of Manufacturing Engineers, Detroit, USA.

Martin 90: Martin P.W. "Linking CAD to CAM for the PCB Industry", Paper available from Automation Ltd. Basingstoke, Hampshire.

Matrox: Matrox Image Series catalogue, available from Matrox Electronic Systems Limited, 1055 St. Regis Boulevard, Dorval, Quebec, Canada.

Matrox pc: Matrox MVP-AT catalogue, available from Matrox Electronic Systems Limited, 1055 St. Regis Boulevard, Dorval, Quebec, Canada.

Matrox sw: Matrox Imager AT software manual, available from Matrox Electronic Systems Limited, 1055 St. Regis Boulevard, Dorval, Quebec, Canada.

Mayhew 92: Mayhew J.E.W. et.al., "The adaptive control of a 4 degrees of freedom stereocamera head", Phil. Trans. of Royal Soc. of London series B-Biological sciences ', 1992,vol 337,no 1281, pp 316-326.

McCabe 76: McCabe T.J., "A complexity measure", IEEE Transactions on Software Engineering, Vol se 2, No 4, Dec. 1976.

McCall 77: McCall J.A. et. al., "Concepts and definitions of software quality", factors in software quality, NTIS, vol. 1, 1977.

MCS 93: "Application and information support systems for planning and control in CIM", Final ACME Report, grant no GR/F 69192, available from the SI Group at LUT.

MDC 91: Weston R.H., Edwards J.M., Hodgson A., 1991, "Model Driven CIM", Grant application to ACME/SERC, available from LUT Manf Eng.

MDC 93: "Model Driven CIM: a framework and toolset for the design, implementation and management of open CIM systems" ACME review report, grant no GR/H/22798, available from SI Group at LUT.

Mertins 92: Mertins K. et.al.,'Flexible software systems for production-adequate control", Production Planning and Control, 1992,Vol 3,No 2, 183-198.

Messina 91: Messina G. and Tricomi G., Proposal for a reference model for vision systems for manufacturing environments. ISO/TC 184/SC 1 N117, Doc. No 91/97534.

Meyer 87: Meyer B., "Reusability: The case for object oriented design", IEEE Software, March 1987, p 50-64.

Microsoft 87: Reference manuals for microsoft C version 5.1 compiler and associated tools MS-DOS, Microsoft Corporation USA.

MMS 90a: ISO/IEC 9506-1: 1990, Industrial Automation Systems - Manufacturing Message Specification - Part1: Service definitions.

MMS 90b: ISO/IEC 9506-2: 1990, Industrial Automation Systems - Manufacturing Message Specification - Part2: Protocol Specification.

Moir 89: Moir P.W., "Efficient beyond imagining, CIM and its application for today's industry", John Wiley and Sons, ISBN 0-7458-0554-x.

Morris 87:Morris H.M., "13th Advanced Control Conference Examines CIM", Control Engineering Journal, July 1987, 75-77.

Munton 91: Munton A., "A vendors perspective of CIME and CIME research", Seminar on CIME, Dept. of Manf Eng LUT, 1991.

Murray 93: Murray R., Fellow of the institute of development studies, University of Sussex, "How what was good for Ford turned bad", The Guardian, p 27, 13 Feb. 93.

NATO 92: Hanke T., Schmid F., Schulz K., "The anthropocentric approach to computer integrated production systems and organisations", NATO Advanced Study Institute, Sept. 1992, Brunel University, UK.

NCR 89: Conversation between the author and Mr. David Walker of NCR, South Carolina USA. during preparation of reference [Edwards 90] April 1989.

Noesis 92: "Visilog: the image processing software", Noesis SARL, Centre d'Affaires de Jouy, 5 bis, Rue du Petit Robinson, 78350 JOUY-EN-JOSAS, France.

Nudd 92: Nudd G.R. et.al., "An heterogeneous M-SIMD architecture for Kalman Filter controlled processing of image sequences", Proc Int. Conf. on computer vision and pattern recognition, Illinios, June 1992.

Nudd 89: Nudd G.R., Francis N.D., "Architectures for image analysis", Proc of IEE conf on Image processing and its applications, Warwick University, July 1989.

Optotrec 89: Image Line, Vision 206,107 catalogue, Available from Optotrec S.A. Brussels, Belgium.

Oughton 88: Oughton D.M.A., "Automated placement of fine lead pitch surface mount electronic devices using vision guidance", Adv. Manf. Engineering Journal, 1988,1,6-10.

OU 92: Open University video on Mechatronics, T395/VCR 1, HOU7193, 26-06-1992.

Owen 87: Owen, J., Bloor, J.S., "Neutral formats for product data exchange: the current situation", 1987, Computer aided design, 19, 436 - 443.

Palfremen 90: Palfremen D., 1990, "converting a black box into a black box", Data Exchange, Manufacturing Systems - The Industrial Systems Integration Journal, May-June, 20-23.

Panasonic 92: Panasonic Factory Automation Company, Panasert Series General Catalogue, Jan. 1992, available through Panasonic UK Ltd., Bracknell, Berkshire, UK.

PDES 87: 1987, A report of the PDES initiation activities. Bradford M Smith, Room A353, Bldg 220, National Bureau of Standards, Gaithersburg, MD 20899, USA.

Peck 87: Peck R.W., "Advantages of modularity in Robot Application Software", General Electric Company, Orlando, Florida, 1987.

Perrier 92: Perrier G., forward to "CIM: Principles of Computer-integrated Manufacturing", by Jean-Baptiste Waldner, Wiley, ISBN 0-471-93450-X

Philips: Philips SBIP VME bus system catalogue, available through Philips Export B.V., I&E - Industrial Automation Systems, Building TQll-p, 5600 MD EINDHOVEN, The Netherlands.

Pimentel 90: Pimentel J. R., "Communications networks for manufacturing", Prentice Hall International Editions, 1990.

Powell 89: Powell J., Carignan B., "CAD reference for AOI", Printed Circuit Fabrication Journal, December 1989, 77-93.

Popek 81: Popek G., Walker, Chow, Edwards, Kline, Rudisin, Thiel, "LOCUS: a network transparent high reliability distributed system", ACM operating systems review, 15(5), 169-177,(December 1981).

Pollard 91: Pollard S.B., Porrill J., Mayhew J.E.W., "Recovering partial wire frame descriptions from stereo data", Image and Vision Computing 91, Vol 9 No1, pp 58-65.

Prabhaker 92: Prabhaker S. and VanDruten J., "An evaluation of object oriented design methodologies for control and manufacturing environments", Proc. of Conf. on Object Oriented Manufacturing, Canada, 1992.

Pugh 83: Pugh A., "Robot Vision", IFS Publications,1983,ISBN 0-903608-32-4.

Rajagopalan 92: Rajagopalan R. and MacNeil P.E., "Object oriented design for CIM applications", School of ENG. Mercer University, Macon, Georgia, USA. Proc. Conf Object Oriented Manufacturing, Canada, 1992.

Ranky 86: Ranky P.G, "Computer Integrated Manufacturing: An introduction with case studies", Prentice Hall 1986.

Ralston 87: Ralston D., Munton T., August 1987, "Computer Integrated Manufacturing", Computer-Aided Engineering Journal, 164-174.

Rogers 92: Rogers P., "Object oriented design and control of intelligent manufacturing systems", Proc. of Conf. on Object Oriented Manufacturing, Canada, 1992.

Rycol 91: Rycol M., Pollard S., Brown C., "Multiprocessor 3D vision system for pick and place", Image and vision Computing, vol. 9, No 1, pp33-38.

Schnur 87: Schnur J.A., "Can there be CIM without MRP II", CIM Review, vol. 3,no 4,3-6,1987.

Schofield 88: Schofield N., "CADCAM.opening the doors to CIM", PA Consulting Services Ltd., 1988.

Schneidewind 93: Schneidewind N.F., "New Software Quality Metrics Methodology Standard fills measurement need", Computer,1993, Vol 26, No 4, pp 105-106.

Schildt 87: Schildt H., "C the complete reference", McGraw Hill, 1987, ISBN 0-07-881263-1

Seidewitz 86: Seidewitz E., "Towards a general object oriented software development methodology", Proc. of 1st Int. Conf. on ADA programming language applications for the NASA Space Station, 1986, P. D.4.6.1-D.4.6.14.

Singer 93: Singer S., "fast bus for video data", Image Processing journal, spring 1993, p14-15.

Spackman 92: Spackman, J.,1992, "Openness: An independent view", Keynote address at DIGITAL world tour - The open advantage in action, Digital Equipment Co. Ltd.

Spenley 91: Spenley P., "Total Quality Management", Dec. 1991, Chapman and Hall, ISBN 0-412-36120-5

Steudel 91: Steudel H.J., "Ten steps to world class manufacturing", Van Nostrand Reinhold Inc., ISBN 0-442-00182-7

STEP 89: STEP Preliminary Design Document & Introduction Document. ISO TC184/sc4/wg1, Obtainable from CADETC, Leeds UNI, UK.

Stroebel 89: Stroebel T., "Fine pitch SMT assemblies: emerging inspection issues", Printed Circuit Assembly Journal, 1989,3,21-27.

Stroustrup 86: Stroustrup B., "The C++ Programming Language", Addison-Wesley Publishing Co. 1986, ISBN 0-201-12078-X.

Strom 90: Strom R.E., "HERMES: An integrated language and system for distributed programming", 2nd IEEE workshop on Distributed Systems, 1990.

Sun 90a: Network Programming Guide (manual) revision A March, 1990, SUN Microsystems.

Sun 90b: SunView Programmers Guide (manual) revision A March 1990, SUN Microsystems

Thacker 91: Thacker N.A., Mayhew J.E.W., "Optimal combination of stereo camera calibration from arbitrary stereo images", Image and Vision Computing, Vol9, No1, 1991, pp27-32.

Thomas 91: Thomas R, "Artificial intelligence in Engineering, Chapter 7 - Computer vision in industry", 245-290, Ed. Winstanley G., John Wiley and Sons 1991, ISBN 0-471-92603-5.

Thomson 89: Thomson V.,Graefe U., "CIM - a manufacturing paradigm", I.J. CIM, Vol. 2, N0 5,290-297

Town 91: Town B. "The science of software selection", Image Processing Journal, september/october 91, p22-23.

Tripathi 89: Tripathi A. R., "An overview of the NEXUS distributed operating system design" IEEE Transactions on software design. Vol 15, No 6, June 1989.

Voss 89: Voss W.D., et.al., "Measure solder paste in the third dimension", Electronic Packaging Production Journal, 1989,1,42-44.

Waldener 92: Waldener J.P., "CIM: Principles of Computer-integrated Manufacturing" Wiley 1992, ISBN 0-471-93450-x

Watts 87: Watts R., "Measuring Software Quality", NCC Publications, 1987, ISBN 0-85012-557-x

Wee 87: Wee H.V.D, "Prosperity and upheaval - the world economy 1945-1980",Penguin Books, ISBN 0-14-013604-5

Weindahl 92: Weindahl H.P., "CIM research and development in Germany with special consideration of the know-how transfer to industry", Proc of ICIMS 92, Tsinghua Uni., Beijing, P.R. China.

Wellhoener 88: Wellhoener E-E, "DEC on CIM", "CIM - Mechanical Aspects", Pergamon Infotech LTD, ISBN 0 08 034116 0.

Weston 88: Weston R.H., Gascoigne J.D., Rui A., et al., 1988, "Steps towards information integration in manufacturing", I.J.CIM,Vol 1, No 3, 140 - 153.

Weston 90: Weston R.H., Hodgson A., Coutts I.A., et al., 1990, "Highly extendable CIM systems based on an Integration platform", Proc. CIMCON 90 Conf. NIST. Gaithersburg, Ma, USA.

Weston 91: Weston R.H., 1991, "CIM Integration Toolboxes". Tutorial, Proc.Int.Conf on CIM, ICCIM'91,Singapore,Sept. PP.99-157.

Weston 92: Weston R.H. and Edwards J., "An architecture and toolset for enterprise wide integration", Proc of ICIMS 92, Tsinghua Uni., Beijing, P.R. China.

Wiener 88: Wiener R.S., Pinson L.J., "An introduction to object oriented programming and C++", Addison Wesley 1988, ISBN 0-201-15413-7.

Williamson 92: Williamson J., Milner C., "The World Economy", Harvester Wheatsheaf, ISBN 0-7450-0773-2

Wong 85: Wong D.G., "Digital Systems Design", Edward Arnold, 1985, ISBN 0-7131-3539-5.

Worhach 92: Worhach P., "Object oriented simulation for equipment level design and analysis in semiconductor manufacturing", Proc. of Conf. on Object Oriented Manufacturing, Canada, 1992.

Wright 90: Wright S.R., "The "Syntegration" of the PCB Fab. Shop", PC Fabrication journal. Feb. 1990, 136-140.

Wright 88: Wright P.K. and Bourne, D.A., "Manufacturing intelligence", Addison Wesley, Reading MA, 1988.

Yim 92: Yim P., "CIM 3: Computer Integrated Man-Machine manufacturing systems", Proc of ICIMS 92, Tsinghua Uni., Beijing, P.R. China.

Young 91: Young R.E., Vesterager J., "An approach to CIM system development whereby manufacturing people can design and build their own CIM systems", Int J. CIM, 1991, Vol 4, No 5, 288-299.

Yourdon 79: Yourdon E. and Constantine L.L., "Structured Design: Fundamentals of a discipline of computer program and systems design, Prentice Hall, 1979.

Zhang 92: Zhang J.B., Weston R.H., "Design and implementation of a CAD information supported machine vision inspection system for PCB manufacture". Prospective conference paper available from SI Group, Manf Eng LUT.

Zwem 90: Zwem A., "CIMplementation: AOI-Based process control", PC Fabrication journal, September 1990, 88-91.

Appendix 1

CONTEMPORY GENERAL PURPOSE VISION HARDWARE

1.0 THE PHILIPS SINGLE BOARD IMAGE PROCESSOR

Figure 1 shows the logical breakdown of this single board system [Philips]. The architecture is typical of many single board solutions and is based around the image memory or frame store. Analog to digital conversion of the video input source is followed by input lookup tables for transformations on incoming signals prior to storage. Output lookup tables precede the digital to analog conversion to generate the graphics output signal. The correlator provides high speed binary operations and real time correlation between reference images and the incoming captured image. On board intelligence is provided through the a 32 bit general purpose processor with graphics processing capabilities. Communication with the host system is effected via the VME bus.

2.0 IMAGING TECHNOLOGY SERIES 150

The overall architecture of this system is shown in Figure 2 [Imaging]. The system comprises a range of 6 boards which use the VME bus for communication to their host system plus a range of video buses for inter-board communication of vision data, including a video pipeline data bus. Facilities provided are as follows:

- analog/digital interface for conversion of signals to video monitors and from video sensors (ADI-150);
- frame buffering, (up to 4 frame buffer cards can be used for image storage) (FB-150);
- real time image processing through the use of a high performance pipe line processor for arithmetic, logical and bit-plane oriented image processing (ALU-150);
- real time convolution, 4*4 convolution on a 512*512 frame in 1/30 sec (RTC-150);
- histogram generation and feature extraction (HF-150);

- programmable image processing acceleration for frequency domain filtering, geometric transformation and correlation (IPA-150).

2.1 Further General Purpose Architecture

Other common architecture at this level include systems based on the use of parallel digital signal processors [LSI] and array processors [Analogic]. The common use of video buses within the various system architecture at this level has led to calls for a standard in this area [Singer 93].

FIGURE 1. Architecture of the Philips single board image processor

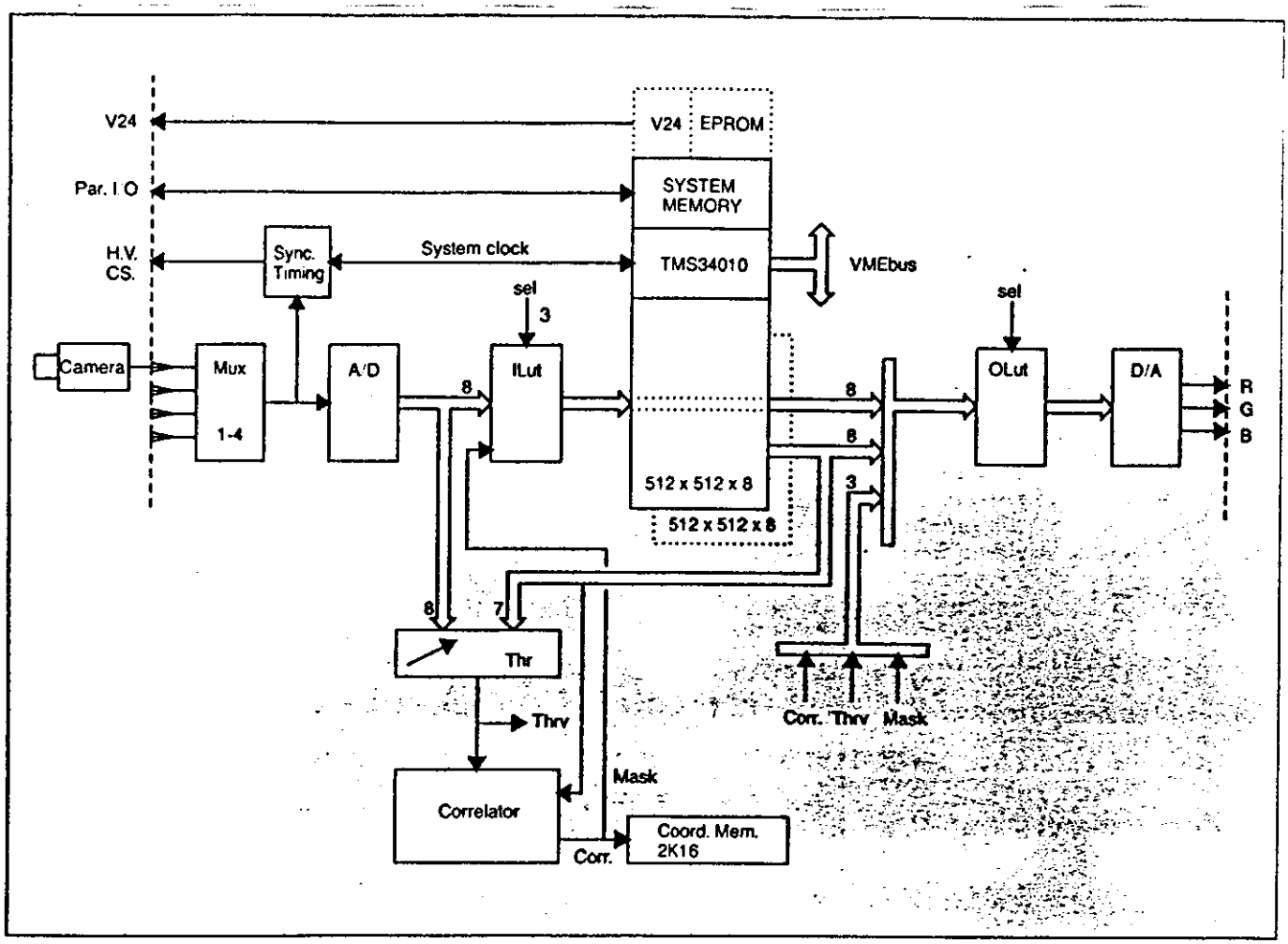
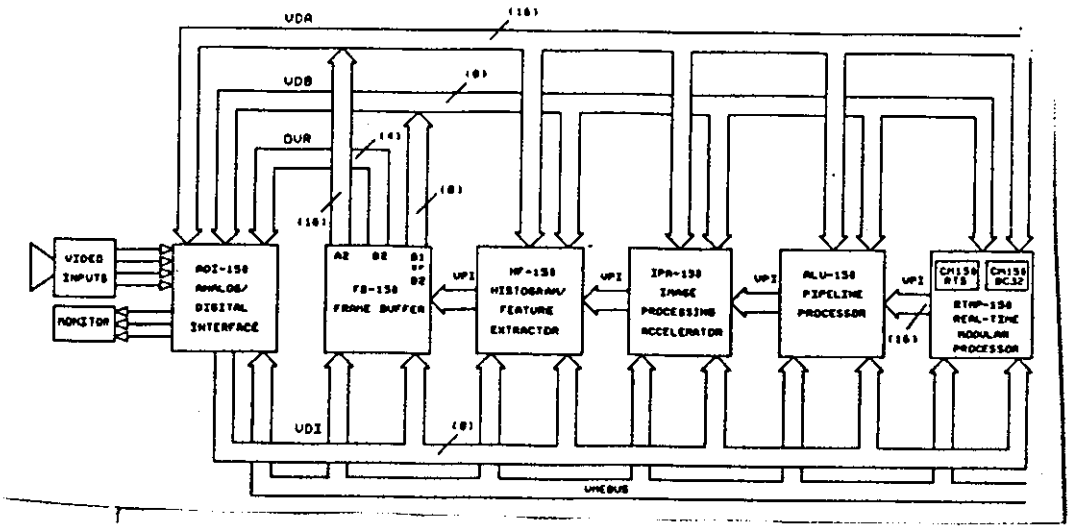


Figure taken from ref. [Philips].

FIGURE 2. Architecture of the Imaging Technology Series 150 system



VDA - video data sourced from framestore A
VDB - video data sourced from framestore B
VDI - video data i/p / feedback
VPI - video pipeline i/p o/p

Figure taken from ref. [Imaging]

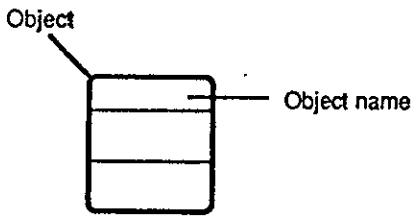
Appendix 2

OBJECT - ORIENTED DIAGRAMING NOTATION

Coad Yourdon Notation

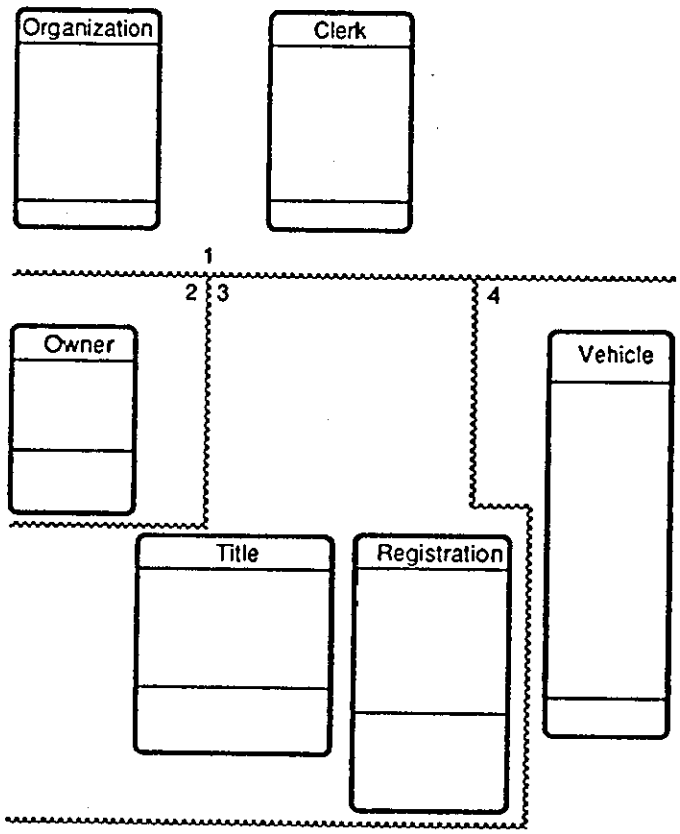
Summarizing the "Identifying Objects" step:

Notation



Strategy

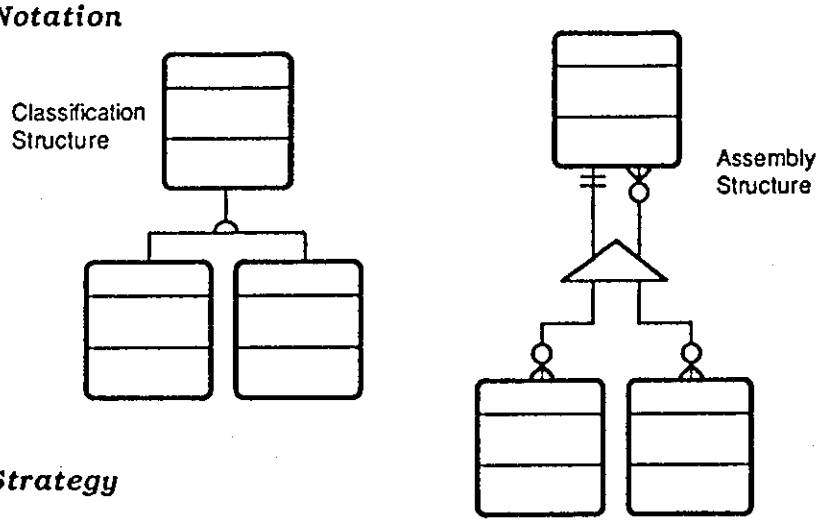
Object = an abstraction of data and exclusive processing on that data, reflecting the capabilities of a system to keep information about or interact with something in the real world.



An OOA Example: Object Layer

Summarizing the "Identifying Structure" step:

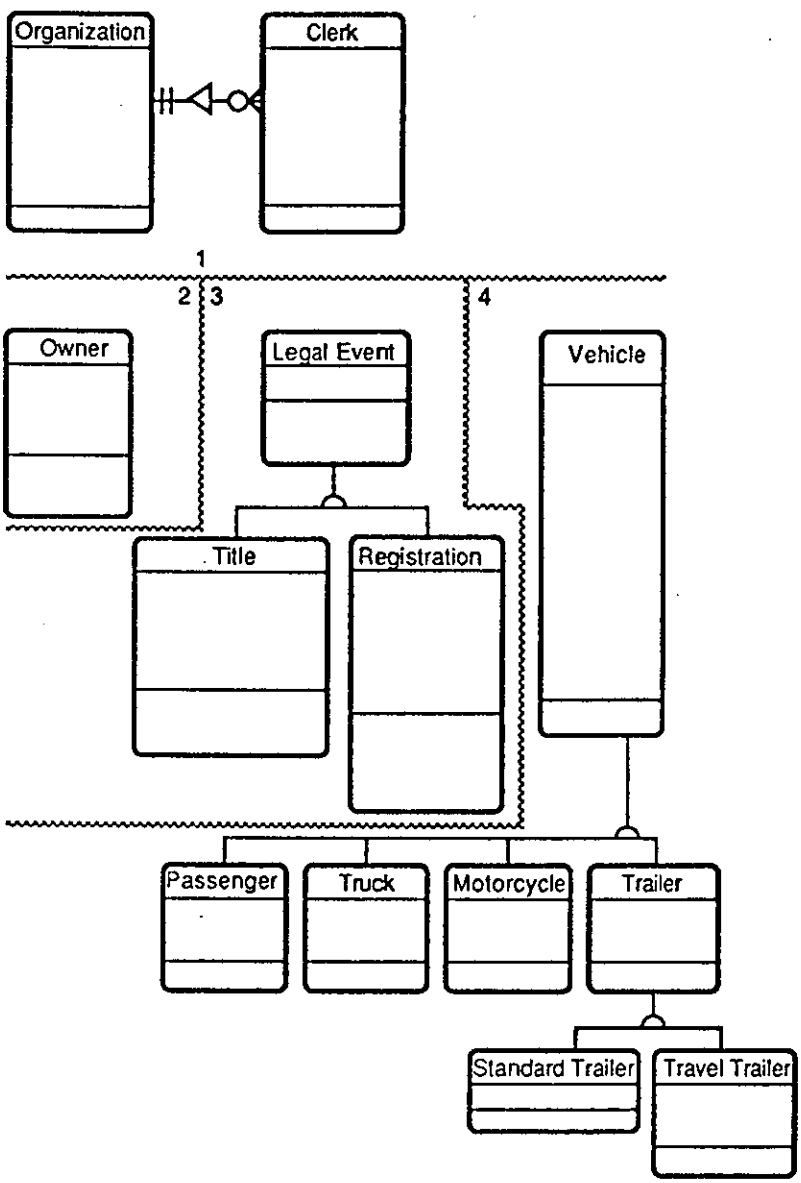
Notation



Strategy

Structure = representation of complexity in a problem space.

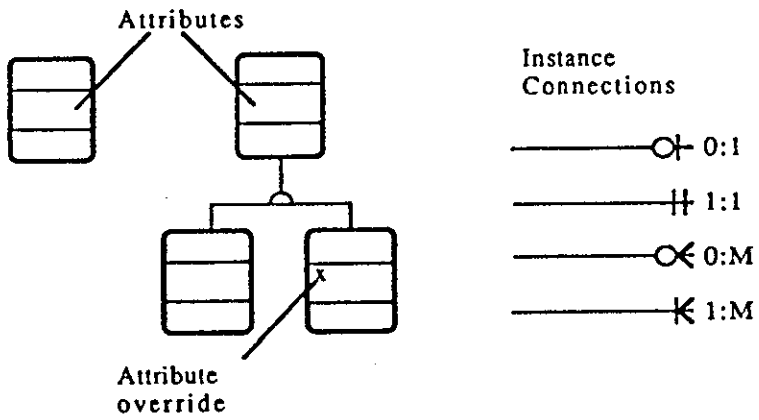
Classification Structure represents class-member organization, reflecting generalization-specialization. Assembly Structure represents aggregation, reflecting whole and component parts.



An OOA Example: Structure Layer

Summarizing the "Defining Attributes" step:

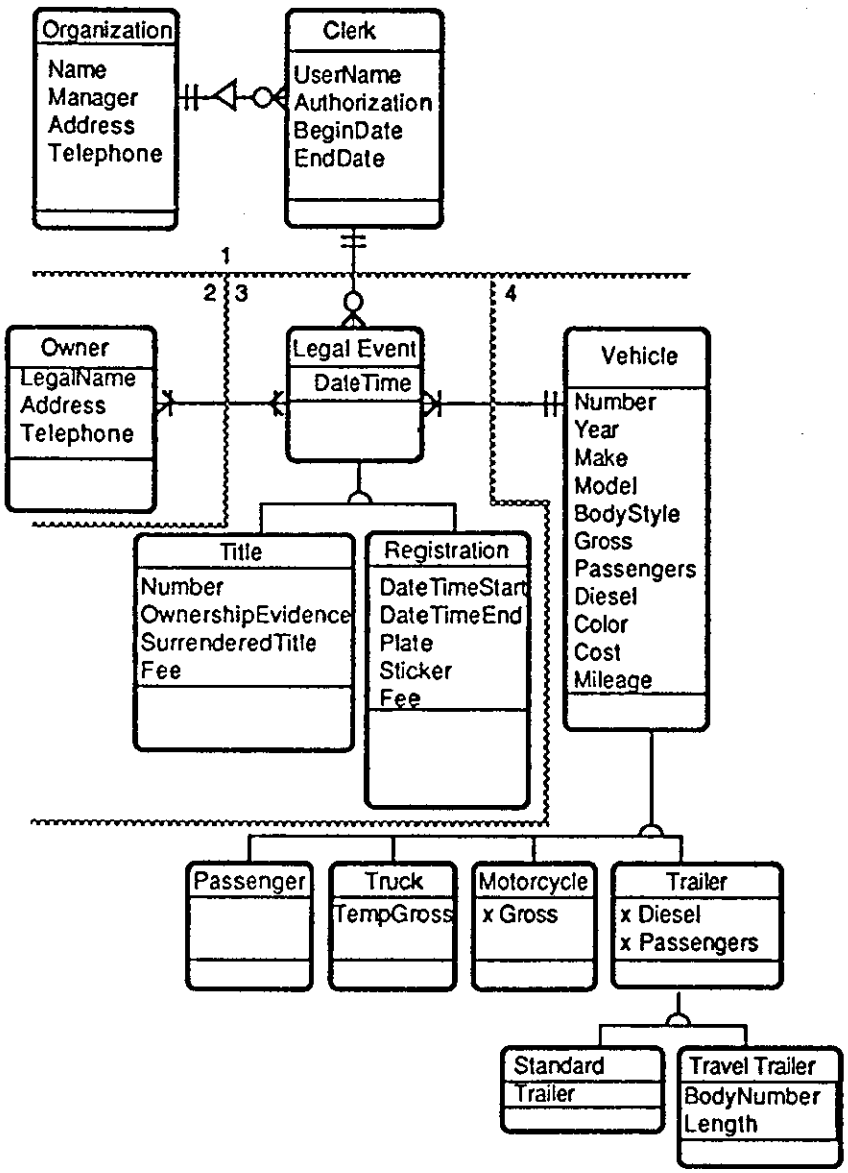
Notation



Strategy

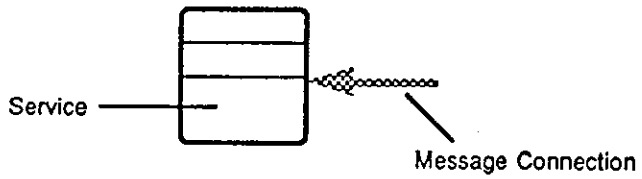
Attribute = a data element used to describe an instance of an Object or Classification Structure.

Attribute Layer:



Summarizing the "Defining Services" step:

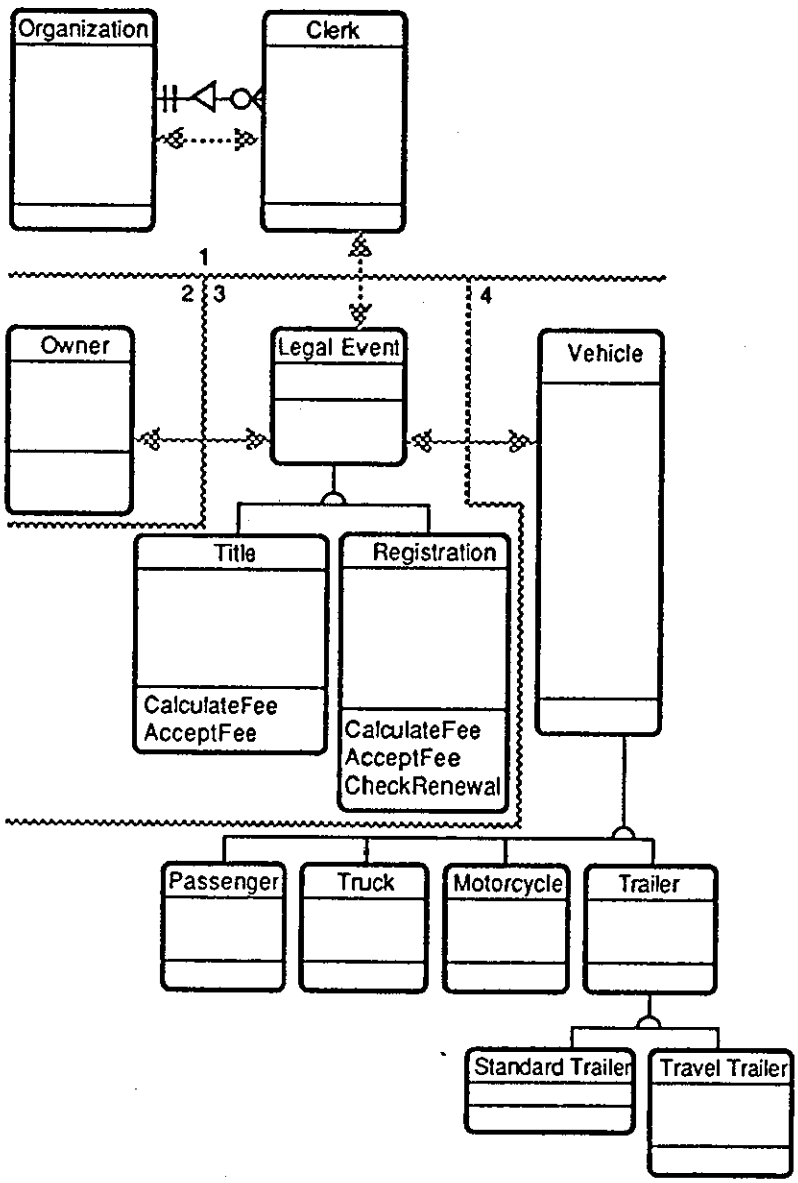
Notation



Strategy

Service = the processing to be performed upon receipt of a message.

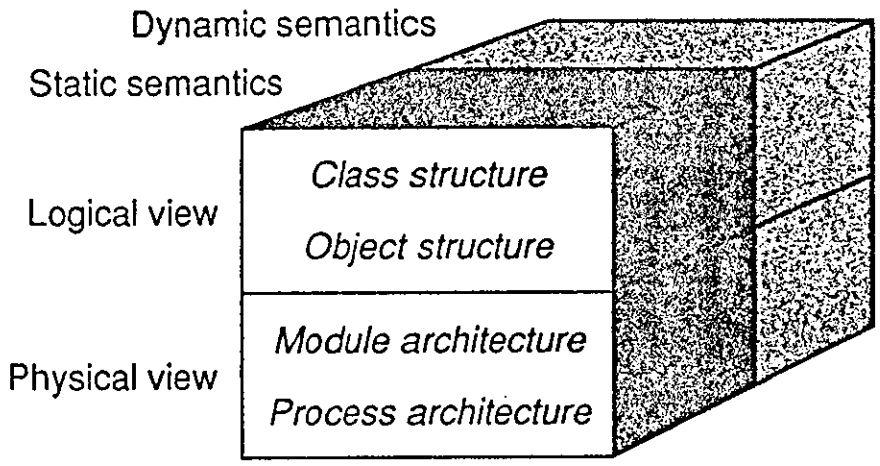
Service Layer:



Booch 91 notation

The Models of Object-Oriented Design

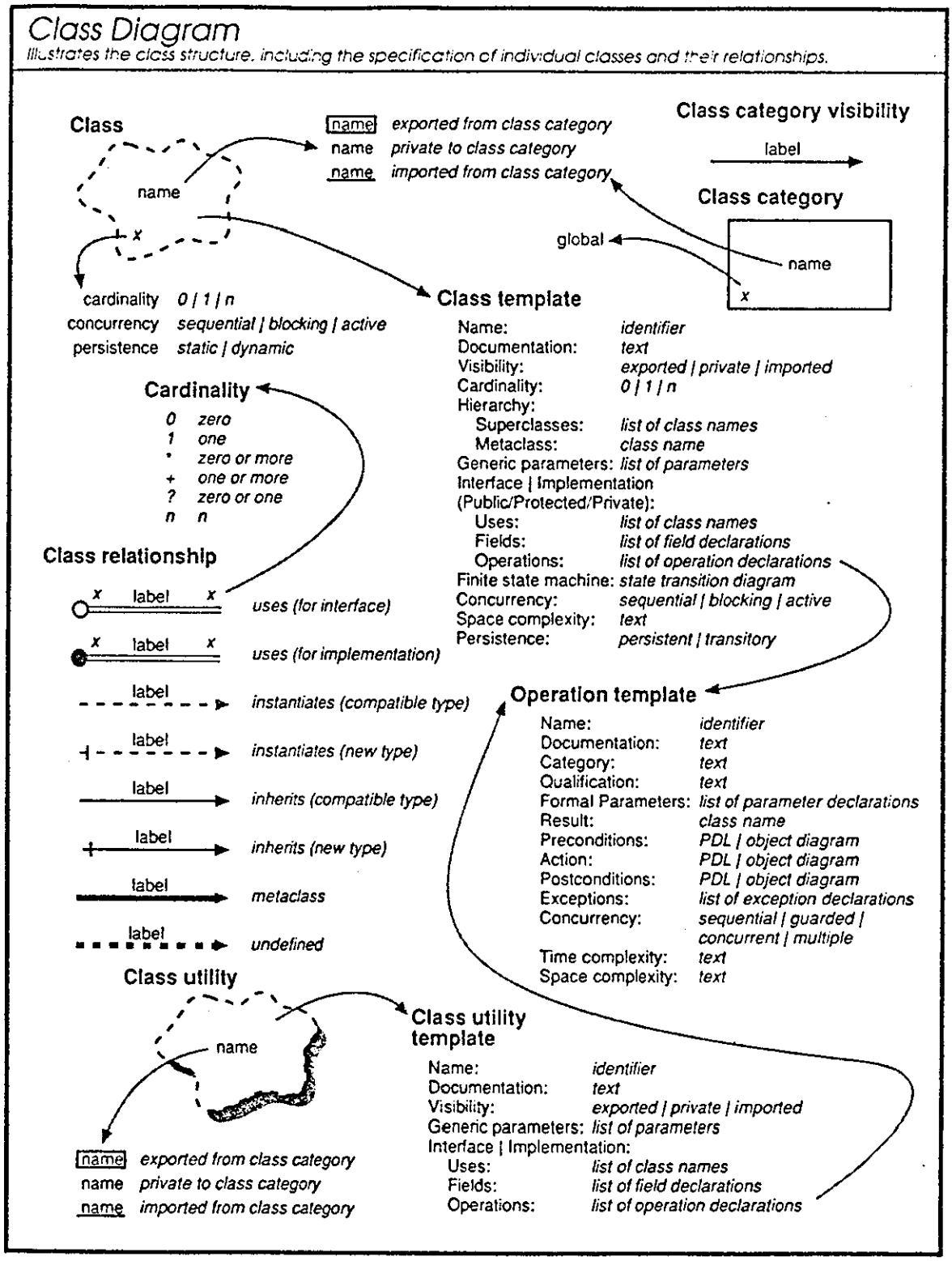
Supporting multiple, interrelated views



The Process of Object-Oriented Design

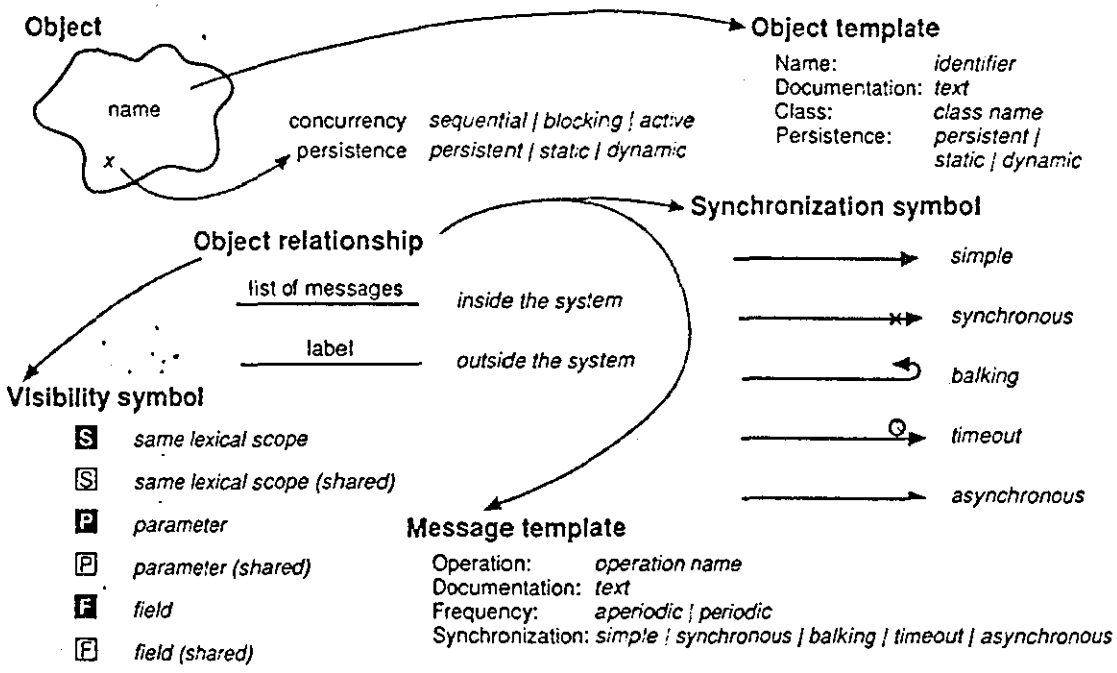
Supporting the incremental and iterative process of round-trip gestalt design

- Identify the classes and objects at a given level of abstraction
- Identify the semantics of these classes and objects
- Identify the relationships among these classes and objects
- Implement these classes and objects


Class Diagram
illustrates the class structure, including the specification of individual classes and their relationships.
Class
name
x
cardinality 0 | 1 | n
concurrency sequential | blocking | active
persistence static | dynamic
name exported from class category
name private to class category
name imported from class category
Class category visibility
label
Class category
global
name
x
Class template
Name: identifier
Documentation: text
Visibility: exported | private | imported
Cardinality: 0 | 1 | n
Hierarchy:
Superclasses: list of class names
Metaclass: class name
Generic parameters: list of parameters
Interface | Implementation (Public/Protected/Private):
Uses: list of class names
Fields: list of field declarations
Operations: list of operation declarations
Finite state machine: state transition diagram
Concurrency: sequential | blocking | active
Space complexity: text
Persistence: persistent | transitory
Cardinality
0 zero
1 one
* zero or more
+ one or more
? zero or one
n n
Class relationship
x label x uses (for interface)
x label x uses (for implementation)
label instantiates (compatible type)
label instantiates (new type)
label inherits (compatible type)
label inherits (new type)
label metaclass
label undefined
Operation template
Name: identifier
Documentation: text
Category: text
Qualification: text
Formal Parameters: list of parameter declarations
Result: class name
Preconditions: PDL | object diagram
Action: PDL | object diagram
Postconditions: PDL | object diagram
Exceptions: list of exception declarations
Concurrency: sequential | guarded | concurrent | multiple
Time complexity: text
Space complexity: text
Class utility
name
name exported from class category
name private to class category
name imported from class category
Class utility template
Name: identifier
Documentation: text
Visibility: exported | private | imported
Generic parameters: list of parameters
Interface | Implementation:
Uses: list of class names
Fields: list of field declarations
Operations: list of operation declarations

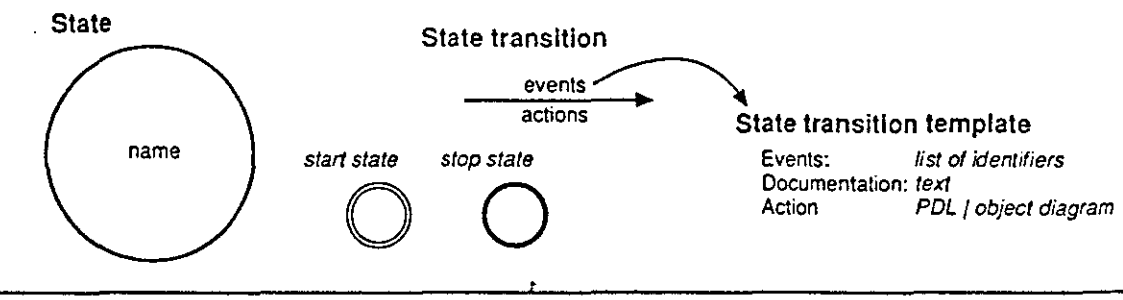
Object Diagram

Illustrates the object structure, including the specification of individual objects and their relationships.



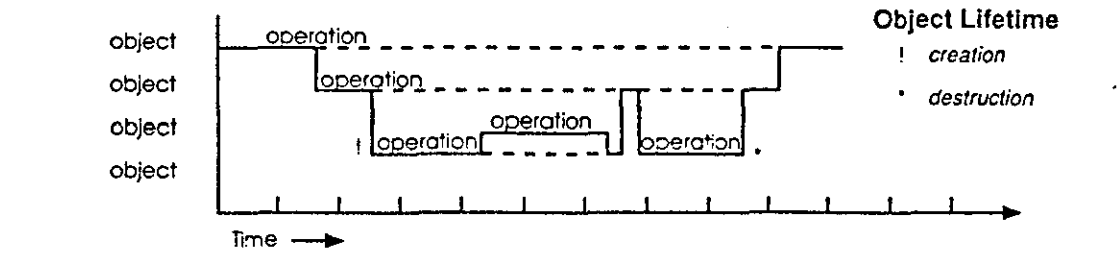
State Transition Diagram

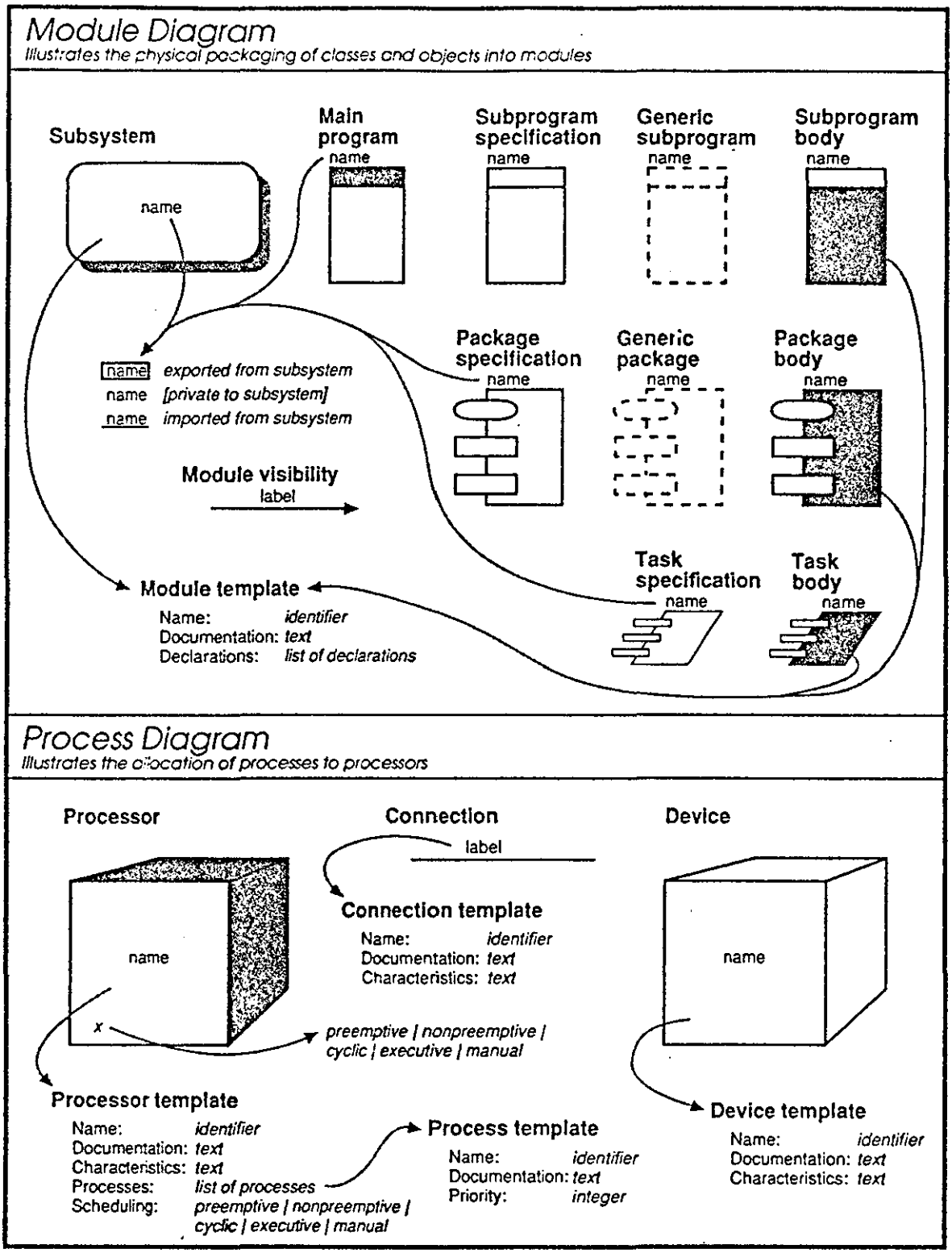
Part of a class diagram, illustrates the state machine of a class



Timing Diagram

Part of an object diagram, illustrates the order of events among a set of objects




Module Diagram
Illustrates the physical packaging of classes and objects into modules
Subsystem
name
Main program
name
Subprogram specification
name
Generic subprogram
name
Subprogram body
name
name exported from subsystem
name [private to subsystem]
name imported from subsystem
Package specification
name
Generic package
name
Package body
name
Module visibility
label
Module template
Name: identifier
Documentation: text
Declarations: list of declarations
Task specification
name
Task body
name
Process Diagram
Illustrates the allocation of processes to processors
Processor
name
x
Connection
label
Connection template
Name: identifier
Documentation: text
Characteristics: text
preemptive | nonpreemptive | cyclic | executive | manual
Device
name
Processor template
Name: identifier
Documentation: text
Characteristics: text
Processes: list of processes
Scheduling: preemptive | nonpreemptive | cyclic | executive | manual
Process template
Name: identifier
Documentation: text
Priority: integer
Device template
Name: identifier
Documentation: text
Characteristics: text

Appendix 3

EXTRACTS FROM THE EDIF CONCEPTUAL MODEL

Entities Relevant To The Inspection Of Bare PCB's

```
ENTITY layout
    SUPERTYPE OF
        (ONEOF (bare_board, test_point, tooling_place, mounting_place,
                sub_layout, flat_footprint, toeprint, heelprint));

    has_layout_primitives : SET [0 : #] OF layout_primitive;
WHERE
    only_occurs_in_relation_to :
        valid_users (LAYOUT,
            ['BARE_BOARD.HAS_TEST_POINTS',
            'BARE_BOARD.HAS_TOOLING_PLACES',
            'BARE_BOARD.HAS_MOUNTING_PLACES',
            'ASSEMBLED_BOARD.IS_BASED_ON',
            'BARE_BOARD.HAS_SUB_LAYOUTS',
            'SUB_LAYOUT.HAS_SUB_LAYOUTS',
            'BARE_BOARD.HAS_FOOTPRINTS',
            'SUB_LAYOUT.HAS_FOOTPRINTS',
            'STRUCTURED_FOOTPRINT.HAS_TOEPRINTS',
            'STRUCTURED_FOOTPRINT.HAS_HEELPRINT',
            'COMPONENT.REFERS_TO',
            'ELECTRICAL_COMPONENT_TERMINAL.CONNECTS_TO_TOEPRINT'
            'COMPONENT_GROUP.BELONGS_TO']);

    can_not_exist_alone :
        can_exist_alone (LAYOUT, FALSE);

    depends_upon_one_of :
        used_by_one_of (LAYOUT,
            ['BARE_BOARD.HAS_TEST_POINTS',
            'BARE_BOARD.HAS_TOOLING_PLACES',
            'BARE_BOARD.HAS_MOUNTING_PLACES',
            'ASSEMBLED_BOARD.IS_BASED_ON',
            'BARE_BOARD.HAS_SUB_LAYOUTS',
            'SUB_LAYOUT.HAS_SUB_LAYOUTS',
            'BARE_BOARD.HAS_FOOTPRINTS',
            'SUB_LAYOUT.HAS_FOOTPRINTS',
            'STRUCTURED_FOOTPRINT.HAS_TOEPRINTS',
            'STRUCTURED_FOOTPRINT.HAS_HEELPRINT']);
END_ENTITY;
```

```
ENTITY test_point
    SUBTYPE OF (layout);
DERIVE
    the_bare_board : bare_board :=
        back_ref (TEST_POINT,
                  'BARE_BOARD.HAS_TEST_POINTS');
WHERE
    only_occurs_in_relation_to :
        valid_users (TEST_POINT,
                  ['BARE_BOARD.HAS_TEST_POINTS']);

    can_not_exist_alone :
        can_exist_alone (TEST_POINT, FALSE);

    depends_upon_one :
        used_by_one (TEST_POINT,
                  'BARE_BOARD.HAS_TEST_POINTS');
END_ENTITY;


ENTITY layout_primitive
    SUPERTYPE OF (ONEOF (shape, layout_text, trace, pad, padstack, hole));
DERIVE
    the_bare_board : bare_board :=
        find_bare_board (LAYOUT_PRIMITIVE,
                  ['LAYOUT.HAS_LAYOUT_PRIMITIVES']);
WHERE
    only_occurs_in_relation_to :
        valid_users (LAYOUT_PRIMITIVE,
            ['LAYOUT.HAS_LAYOUT_PRIMITIVES',
            'PHYSICAL_NET.HAS_SHAPES',
            'PHYSICAL_NET.HAS_ASSOCIATED_TEXT',
            'PHYSICAL_NET.HAS_TRACES',
            'PHYSICAL_NET.HAS_PADS',
            'PHYSICAL_NET.HAS_EQUIPOTENTIAL_PADSTACKS',
            'PHYSICAL_NET.HAS_PLATED_THROUGH_HOLES',
            'PROBE_POINT.REFERS_TO_LAYOUT_PRIMITIVE',
            'FIDUCIAL_MARK.REFERS_TO_LAYOUT_PRIMITIVES',
            'GEOMETRIC_LAYOUT_TEXT.USES_SHAPES',
            'GEOMETRIC_LAYOUT_TEXT.USES_TRACES',
            'PADSTACK.HAS_PADS'
            'PADSTACK.HAS_HOLES',
            'JUMPER_PADSTACK.REFERS_TO',
            'ELECTRICAL_COMPONENT_TERMINAL.CONNECTS_TO_PIN_PLACE']);

    can_not_exist_alone :
        can_exist_alone (LAYOUT_PRIMITIVE, FALSE);

    depends_upon_one :
        used_by_one (LAYOUT_PRIMITIVE,
            'LAYOUT.HAS_LAYOUT_PRIMITIVES');
END_ENTITY;
```

```
ENTITY pad
      SUBTYPE OF (layout_primitive);

      has_geometry : SET [1 : #] OF geometry;
WHERE
      only_occurs_in_relation_to :
         valid_users (PAD,
            ['PHYSICAL_NET.HAS_PADS',
            'PADSTACK.HAS_PADS',
            'LAYOUT.HAS_LAYOUT_PRIMITIVES',
            'PROBE_POINT.REFERS_TO_LAYOUT_PRIMITIVE',
            'FIDUCIAL_MARK.REFERS_TO_LAYOUT_PRIMITIVES']);

      can_not_exist_alone :
         can_exist_alone (PAD, FALSE);

      depends_upon_one_of :
         used_by_one_of (PAD,
            ['PADSTACK.HAS_PADS',
            'LAYOUT.HAS_LAYOUT_PRIMITIVES']);

(*
      The geometry must use physical layers which the pad can go on
*)

      r1 :
         is_empty ({g <* has_geometry |
         NOT g.used_physical_layers <=
            the_bare_board.is_implemented_in.has_physical_layers});
END_ENTITY;


ENTITY hole
      SUBTYPE OF (layout_primitive)
      SUPERTYPE OF (ONEOF (plated_through_hole, unplated_hole));

      spans : SET [1 : #] OF physical_layer;
      has_geometry : SET [1 : #] OF geometry;
WHERE
      only_occurs_in_relation_to :
         valid_users (HOLE,
            ['PADSTACK.HAS_HOLES',
            'LAYOUT.HAS_LAYOUT_PRIMITIVES',
            'PROBE_POINT.REFERS_TO_LAYOUT_PRIMITIVE',
            'FIDUCIAL_MARK.REFERS_TO_LAYOUT_PRIMITIVES',
            'PHYSICAL_NET.HAS_PLATED_THRCUGH_HOLES']);

      can_not_exist_alone :
         can_exist_alone (HOLE, FALSE);
```



```
ENTITY shape
      SUBTYPE OF (layout_primitive);

      has_geometry : SET [1 : #] OF geometry;
WHERE
      only_occurs_in_relation_to :
         valid_users (SHAPE,
            ['LAYOUT.HAS_LAYOUT_PRIMITIVES',
            'PHYSICAL_NET.HAS_SHAPES',
            'GEOMETRIC_LAYOUT_TEXT.USES_SHAPES',
            'PROBE_POINT.REFERS_TO_LAYOUT_PRIMITIVE',
            'FIDUCIAL_MARK.REFERS_TO_LAYOUT_PRIMITIVES']);

      can_not_exist_alone :
         can_exist_alone (SHAPE, FALSE);

      depends_upon_one :
         used_by_one (SHAPE, 'LAYOUT.HAS_LAYOUT_PRIMITIVES');

(*
      The geometry must use physical layers which the shape can go on
*)

      r1 :
         is_empty ({g <* has_geometry |
         NOT g.used_physical_layers <=
            the_bare_board.is_implemented_in.has_physical_layers});
END_ENTITY;


ENTITY trace
      SUBTYPE OF (layout_primitive);

      has_geometry : SET [1 : #] OF geometry;
      trace_width : INTERNAL DISTANCE;
      end_type : INTERNAL END_TYPE;
      corner_type : INTERNAL CORNER_TYPE;
      status : FIXED_INDICATOR;
DERIVE
      true_status : FIXED_INDICATOR := status_to_be_written;
WHERE
      only_occurs_in_relation_to :
         valid_users (TRACE,
            ['PHYSICAL_NET.HAS_TRACES',
            'GEOMETRIC_LAYOUT_TEXT.USES_TRACES',
            'LAYOUT.HAS_LAYOUT_PRIMITIVES',
            'PROBE_POINT.REFERS_TO_LAYOUT_PRIMITIVE',
            'FIDUCIAL_MARK.REFERS_TO_LAYOUT_PRIMITIVES']);

      can_not_exist_alone :
         can_exist_alone (TRACE, FALSE);
```

Appendix 4

AN EMERGENT MULTI-COMPONENT / MULTI-LAYER MODEL BASED ON OBJECT-ORIENTATION.

The structure of software within the application layer of the architecture proposed within this thesis cannot be truly generic in nature, due to various demands of the many different forms of machine vision application that are possible. However, domain specific structure, (or partial models [CIM-OSA]) can be used to promote a standard framework within the application layer software.

A general structure derived from the "EDIF model for PCB" [EDIF 90, 91], was identified by the author in which four layers were used to structure the problem domain component as shown in Figure 5 of chapter 7. The principal objects within the application were also derived from the EDIF model as was the naming convention used within the application (these being directly related to object names within the EDIF model). This notion of using application domain models to identify principal objects, and thereby generally influence design decisions within the application, was derived from the consistent occurrence of structure in each application implemented by the author. All applications built by the author have adopted a structure which reflects the physical structure of the object to be inspected. It is this phenomena which forms the basis of object oriented analysis. Existing models (such as EDIF) that have been derived, tested, proved, accepted by, and are used within industry are a sensible place to search for objects which can contribute to the data abstractions used within an application. The use of such technique can lend industry wide familiarity to the form of applications.

Decomposition within the application layer supported the notion of the 4 component model described by Coad in his paper "new advances in object oriented analysis" [Coad 91]. These components being:

- problem domain component;
- operator interface component;

- task management component;
- information management component

Figure 7 in Chapter 7 shows the object decomposition within the vision application object. The "problem domain component" and the "management component" are clearly shown. Operator interface facilities exist within the client application, a clear demonstration of the decomposition in line with Coad's proposals.

Coad suggests the model is multi-layer as well as multi component, identifying the following layers:

- subject layer;
- class and object layer;
- structure layer;
- attribute layer;
- service layer.

The top four layers are analogous to the author's application decomposition, while the services provided by the Vision Model are analogous to Coad's bottom Service layer. The authors proposed decomposition within the applications layer supports the multi-component / multi-layer model seen by Coad as emerging within applications based on the object-orientation.

Appendix 5

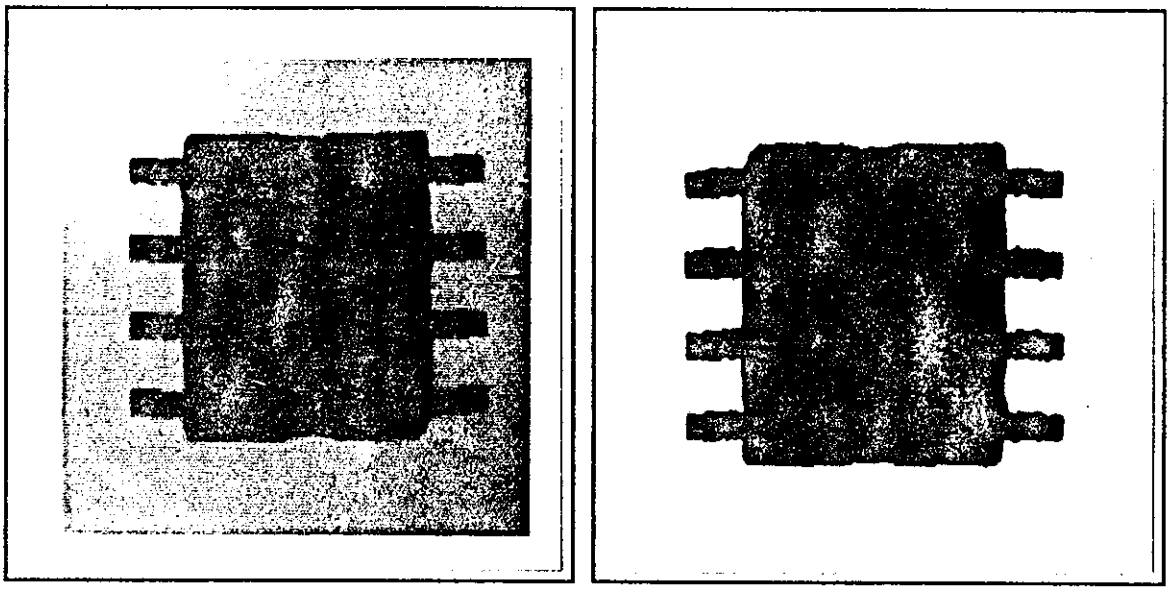
THE APPLICATIONS DURING EXECUTION

1.0 INSPECTION OF THE PINS OF AN SO8

Figure 1 shows photographs of an 8 leg small outline integrated circuit (SO8) under inspection. Figure 1 a shows the back illuminated image which forms the raw image snapped into the vision system frame buffer. Figure 1 b shows the grey level processed and thresholded binary image, while Figure 1 c shows the segmented binary image following noise removal via erosion and dilation, followed by edge detection and thinning, to produce a single pixel boundary. Figure 1 c also show the centroid identification which takes place (together with the computation of area and perimeter) during the extraction of the component boundary.

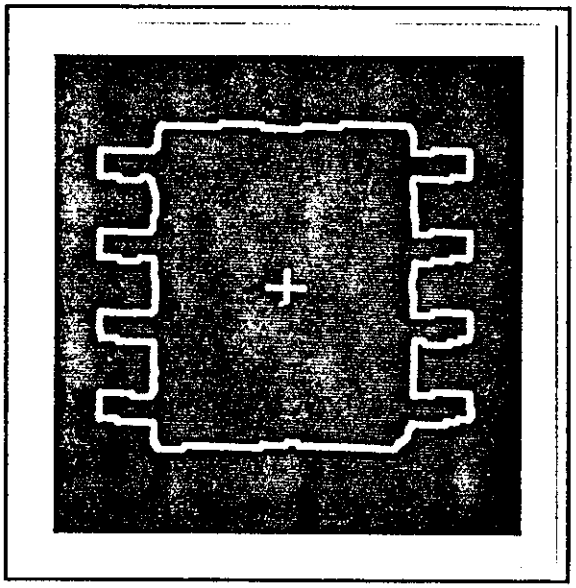
If a classification match is found, the EDIF model Package entity of the particular matched component is loaded from disc. Window co-ordinates for each pin are computed and used to control local vision processing of each pin of the component. The view of the EDIF model information required to drive the functionality implementing the extraction of detailed pin information is achieved by the instantiation and use of information management objects. Figure 2 a shows the result of window processing around pin 1 of the component, following the generation of a single pixel width arc describing the pin. Further sub-windows are computed from the EDIF model package entity to identify regions of prospective corner points. Figure 2 b shows the processing windows, while Figure 2 c shows the identified corners. Figure 2 d shows the component following complete extraction of the pin corner points.

FIGURE 1. Images of the application extracting elementary features



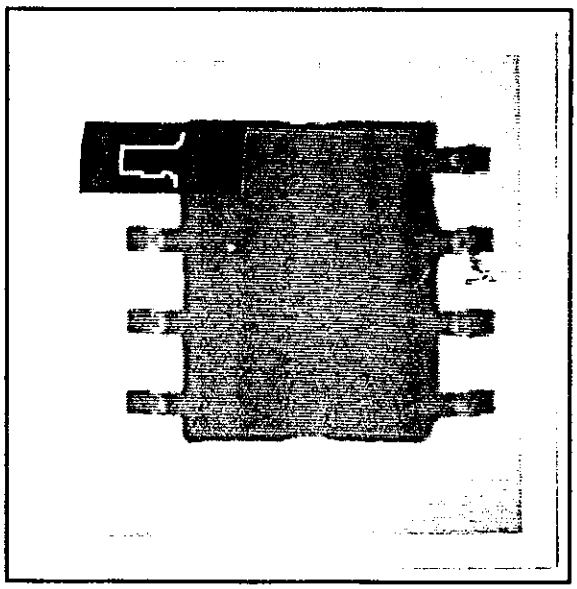
a: raw image of back illuminated component

b:image thresholded to binary

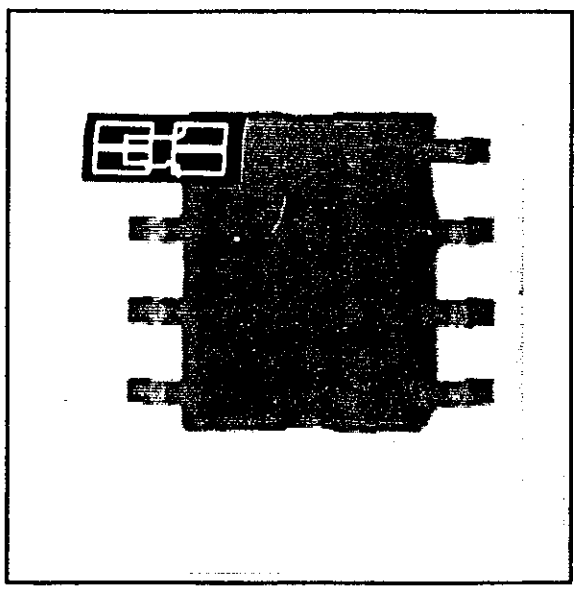


c: segmented binary image following noise removal

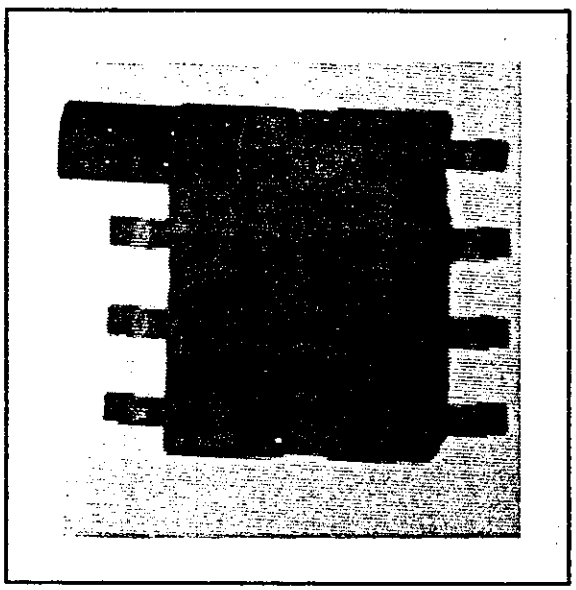
FIGURE 2. Images of the application extracting "live" features from the package pins



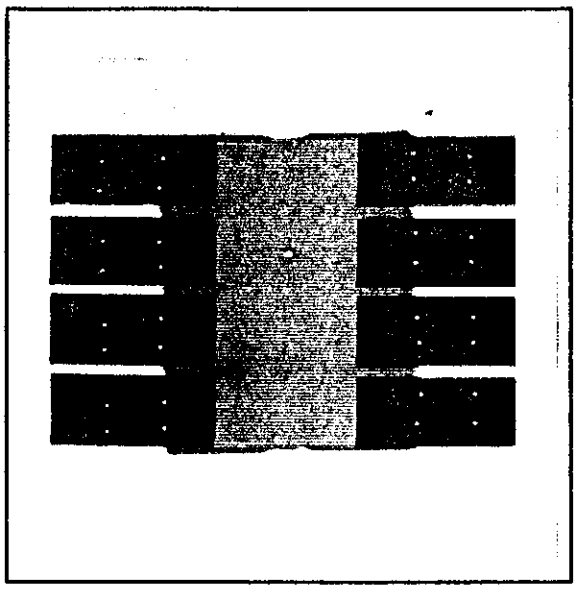
a: window processing around pin 1



b: corner point processing windows



c: identified pin corners

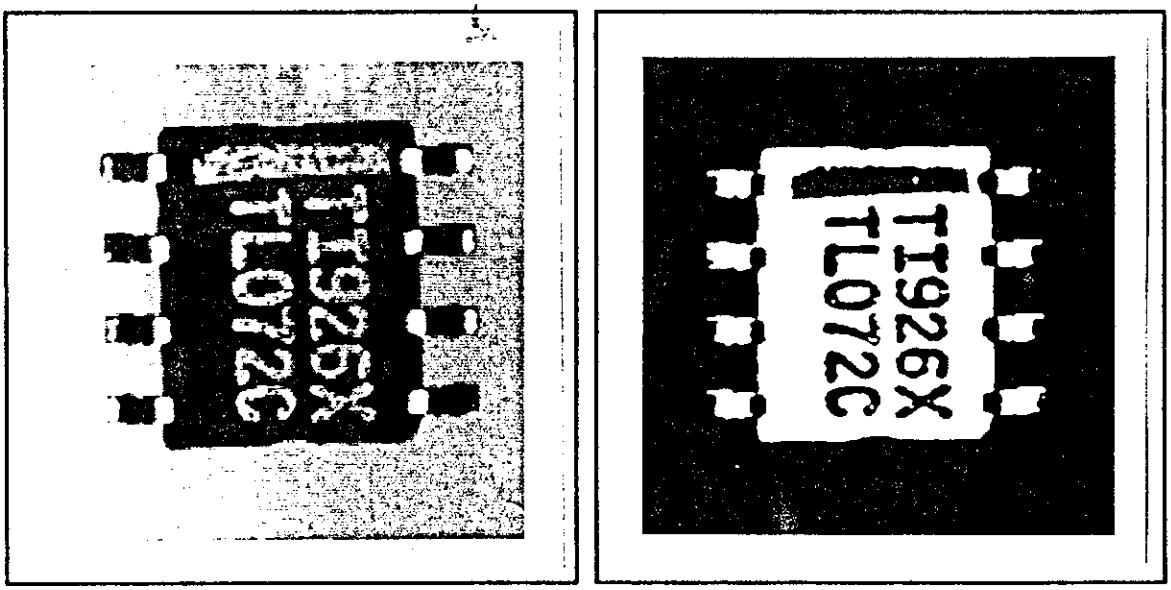


d: extraction of all pin corners

2.0 THE MODIFIED APPLICATION

Figure 3 shows images of the image processing used to generate elementary features from a front lit SO8. The figure shows the final image which produces useful area and perimeter information but cannot be used to extract accurate descriptions of the component Pins. Figure 4 shows images of the processing required to extract descriptions of the component identification. Again window processing based on information extracted from the EDIF product model of the component is used.

FIGURE 3. Images of the modified application extracting elementary features

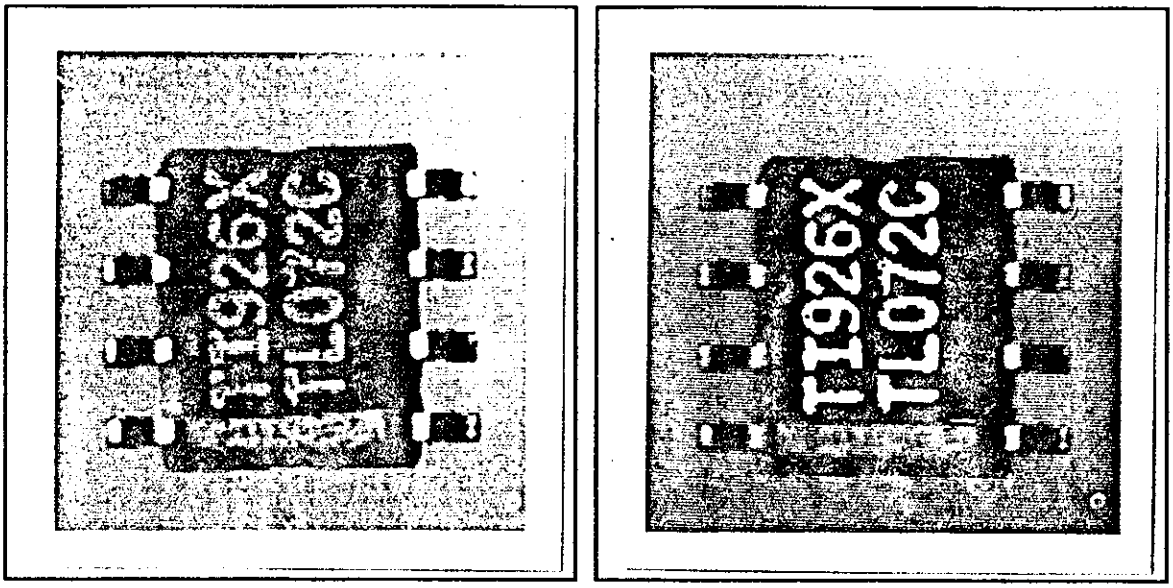


a: raw image of front illuminated component

b: binary level thresholded image

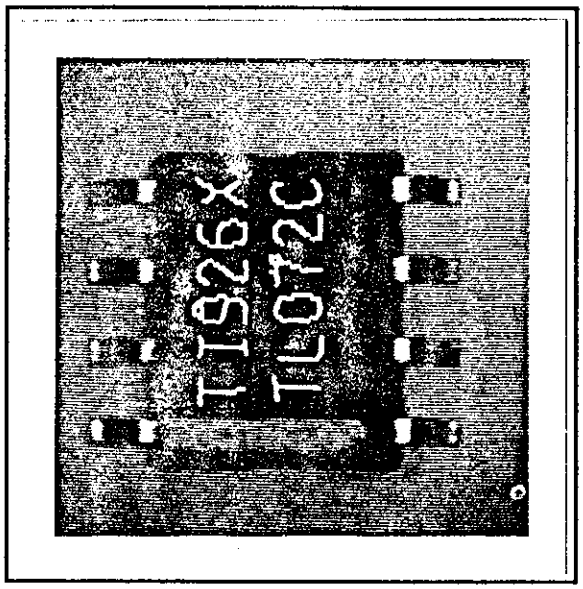
c: segmented binary image following image processing

FIGURE 4. Images of a new application extracting String_Arcs features



a: raw image of front illuminated component

b: binary level thresholded image



c: segmented binary image following image processing

Appendix 6

COST IMPLICATIONS

In his paper on modular application software for robots, Peck [Peck 87] has evaluated the effect of a modular approach on each phase of the software development cycle. Figure 1 gives details of his findings (assuming no re-use of existing software), illustrating the additional time requirements during design, and the reduced time during code generation and implementation. Time during implementation and debug can often tie up valuable resource particularly during production line commissioning and system optimization. Camp [Camp 76] states that a modular approach can reduce implementation costs by between twenty five and fifty per cent. It could be argued that the structured approach proposed within this thesis could reasonably be expected to lead to similar savings.

FIGURE 1. Effects of using a modular approach for application software creation

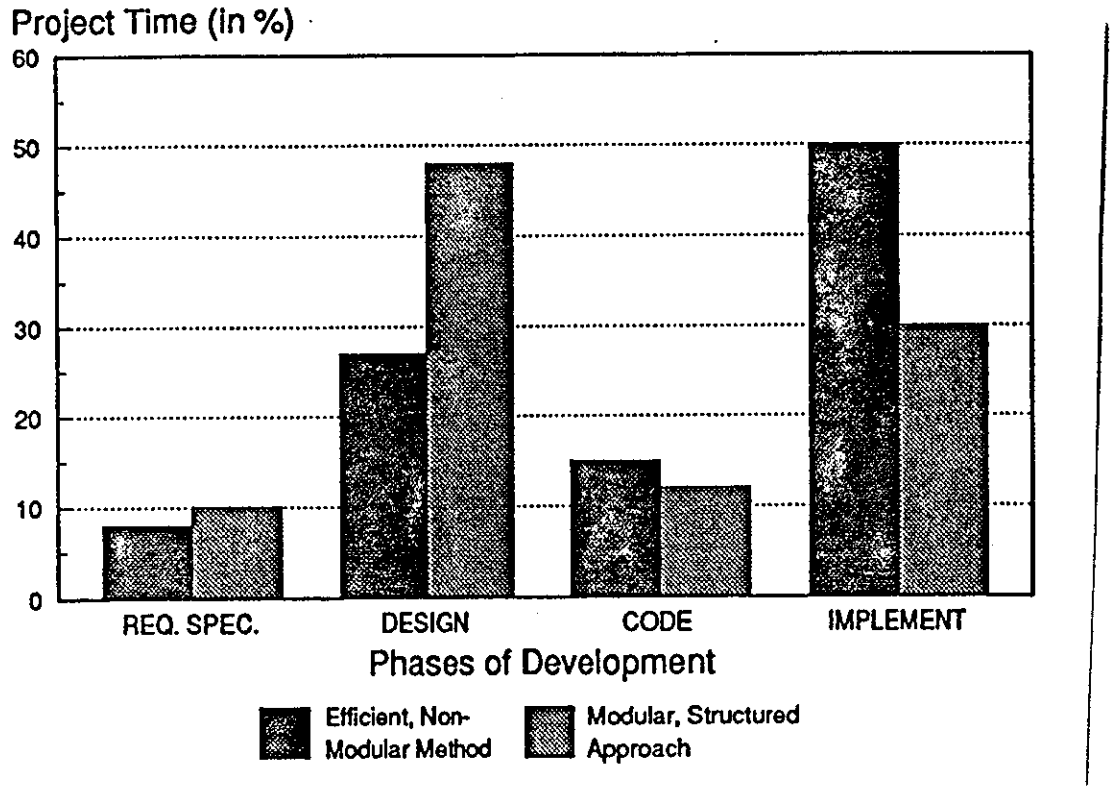


Figure taken from reference [Peck 87]

FIGURE 2. The cost of creating and maintaining application software

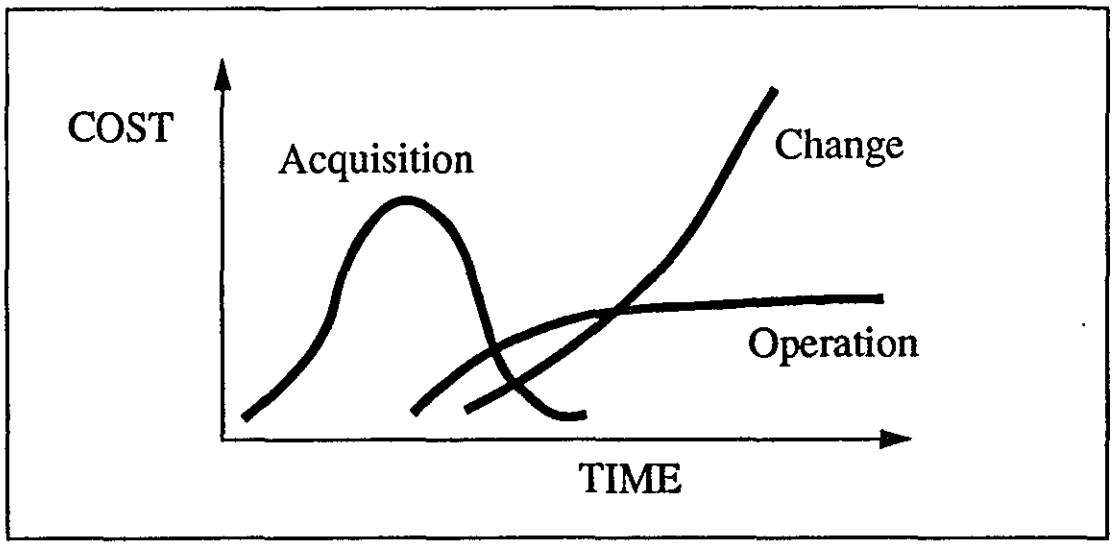


Figure taken from reference [Spackman 92]

The early chapters of this thesis identified the need to cope with change in order to retain a competitive edge gained through the exploitation of contemporary manufacturing practises. Figure 2 shows the trend of cost against time used by Spackman in his paper on open systems integration [Spackman 92]. Spackman states that the modification of next generation open systems will constitute the major cost during a system lifecycle. The proposals made in this thesis to support ease of change within next generation machine vision systems could help to reduce these costs.

Appendix 7

A VISION CLIENT APPLICATION IN A SOFT INTEGRATED VISION MACHINE.

This appendix details a particular vision client application used in the authors implementation of a Soft Integrated Machine Vision System described in PART B and C of this thesis. The functionality provided within this Vision Client Application is designed to fulfil the minimum specification to exercise the Remote Vision Server. The Vision Client Application comprises two principal areas of functionality:

- operator interface, and;
- information management.

Figure 1 and Figure 2 show bit map captured images of the operator interface windows taken from a Sun workstation VDU during operation of the Vision Client Application. The main interface shown in Figure 1 has standard communications monitoring facilities similar to that described for the Vision Alien Device Driver Main Interface. The Top half of the Main Interface has facilities for selecting from a menu of programs for remote invocation (vaple.exe or slow.exe), and offers a field for user entry of any executable program file name which is available within the vision application object.

Control buttons are available on the Main Interface for manual operation of the facilities described within this chapter, typically the establishment of an association with the Vision Alien Device Driver, or the sending of data to the Remote Vision Service Provider.

The Admin. Interface shown in Figure 2 provides application specific facilities and is unique to the particular application. It provides control and display of Information management facilities within the application. A display of Live Model Information provided by the Remote Vision Service Provider is available, with facilities for saving, loading and viewing information pertaining to small outline IC's (SOIC).

FIGURE 1. GUI's for the Vision Client Application

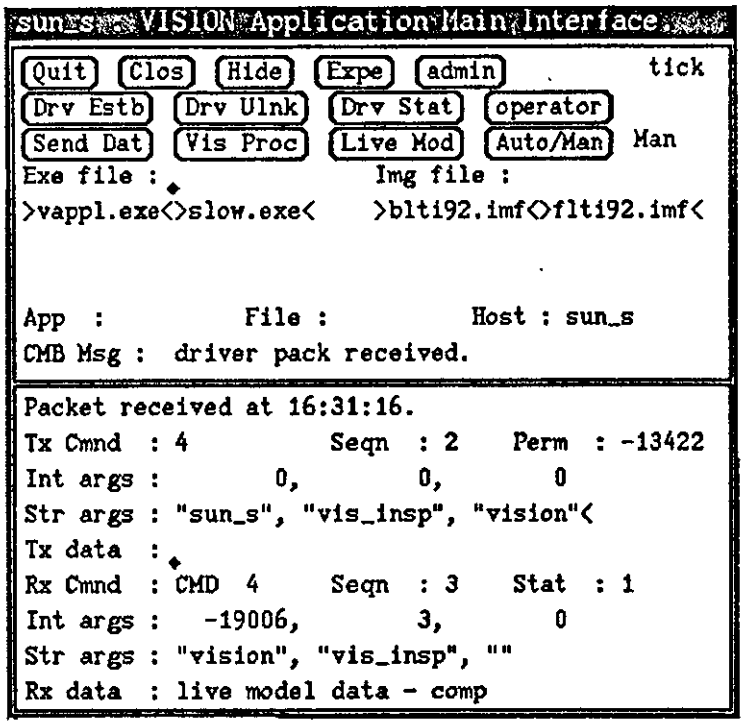
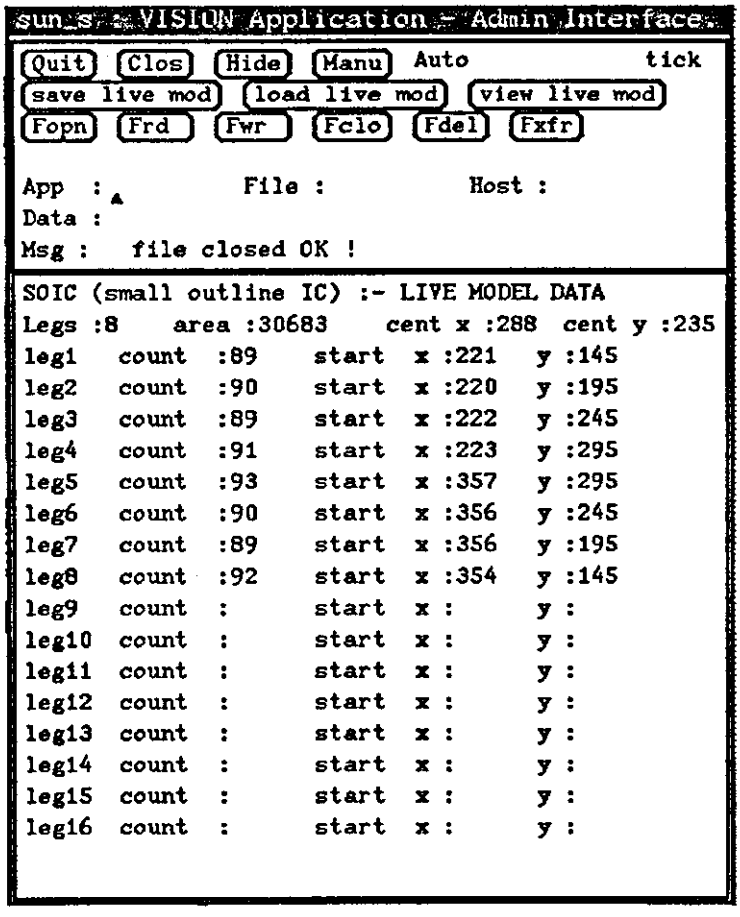


FIGURE 2. GUI for Vision Client Application - Admin. Interface



Appendix 8

PAPERS CONCERNING NEXT GENERATION MACHINE VISION SYSTEMS

INT. J. COMPUTER INTEGRATED MANUFACTURING, 1993, VOL. 6, NO. 5, 323–334

Machine vision integration and information support: methods, models and tools

J. EDWARDS, P. CLEMENTS and S. MURGATROYD

Abstract. In recent years, machine vision systems have been introduced into the electronics manufacturing industry for control and inspection applications. For some applications, great benefit has been derived, but serious limitations affecting flexibility and the system life cycle have been exposed. This paper proposes a model for building integratable vision machines capable of deriving benefit from support information, providing facilities to ease change and improve flexibilily. A proof of concept implementation is discussed using information storage based on both the relational and the object-oriented models. The paper presupposes the need for an integration infrastructure such as that specified by CIM-OSA or as implemented at Loughborough University of Technology by the Systems Integration Group as the CIM-BIOSYS (Computer Integrated Manufacturing Building Integrated Open Systems) integration infrastructure.

1. Introduction

During recent years, vision machines have become an integral part of automated process machines within the electronics manufacturing industry. These processes are being used to increase efficiency and quality levels in the fabrication and assembly of fine-pitch technology printed-circuit boards (PCBs) employing surface mount technology (SMT). Typically they are used as inspection tools during phototool, inner-layer and bare-board fabrication. During PCB assembly they are used in establishing board and component alignment. They are also employed for solder paste and glue-dispensing inspection prior to component placement and at stages of pre- and post-solder inspection.

Automation will only be introduced into the electronics manufacturing industry when it is capable of carrying out a task more accurately, reliably and efficiently than an employee or when the task simply cannot be done manually. However, in today's climate of short product life cycles, short production runs, fast times to market, multiple and custom products produced in a fast and flexible manner, considerations such as the configurability, flexibility, ease of update and change in a manufacturing process machine are additional issues which are key to the successful implementation of automation.

Sakakibara and Matsumoto (1991) in describing their car-electronics product plant (Nippondenso Co., Ltd.) cite the current number of products at 4000, the number of new products as 1000 per year with design changes at 5000 per year. This type of environment makes it very difficult to justify any technology that cannot adapt to change and cannot be easily integrated within an information system. It is information support which is necessary to enable fast product change over and to minimize machine downtime, while reducing the chance of faulty configuration and improving yield. It is these goals then that machine vision process machinery must strive for

(*a*) flexibility, that is the ability to handle multiple predictably change, for example to inspect a range of products,
(*b*) ease of change, that is to support modification in line with unforeseen change, and
(*c*) ease of integration, that is to be a building element within a computer integrated manufacturing (CIM) system.

This paper describes work at Loughborough University of Technology (LUT) aimed at developing the required mechanisms to enable the above requirements to be met. It is these goals of flexibility, ease of change and ease of integration which determine the design and implementation of the system described.

Authors: J. Edwards, P. Clements and S. Murgatroyd, Systems Integration Group, Department of Manufacturing Engineering, Loughborough University of Technology, Loughborough, Leicestershire, LE11 3TU, UK.

0951-192X/93 $10·00

2. Current practice in machine vision systems design

Machine vision inspection machines are primarily complex pieces of mechanical engineering with associated electronic programmable controllers. They are designed with their immediate functionality in mind as standalone processes capably of being bought 'off the shelf'. They can be installed and put into operation without necessarily being linked to complex communications and information systems used in their host environment. They are built to achieve a specific application with configuration facilities to enable multiple products of the same type to be processed. As such, the main design priorities are software execution speed, reliable vision-processing algorithms and mechanical accuracy. Unfortunately when viewed as a building block of a manufacturing system these vision machines are generally not of a flexible nature and as such do not permit a comfortable migration path to enable modification as requirements change, for example with changes in enabling technology, or changing product requirements. Much of their inflexibility stems from a lack of consideration given to the way that such systems may be integrated. Typically, integration provision might comprise a 15-way D-type connector, an RS232 port and a section in a manual specifying the rough requirements to enable communication.

Hence we have the all too familiar sight of the 12-month-old piece of expensive vision machine technology sitting on the shop floor gathering dust, only to be used in the manufacture of the occasional batch of old product.

This paper proposes a vision machine design strategy which can enable the required features of flexibility, ease of change and ease of integration and could prevent the continued generation of inflexible systems. The reduced development and system support costs and the potential increase in the system life cycle of the equipment could also reduce the risk associated with choosing a machine vision option.

3. A structured approach to integration implementation

The Systems Integration (SI) group at LUT has for some years recognized the requirement for a single consistent interface for applications requiring integration level flexibility. This flexibility could be provided through the control and management of interprocess interaction and information sharing, ensuring provision for controlled change. To address this requirement an integration infrastructure and platform of services was derived through practical experience of solving contemporary integration problems. This infrastructure is currently known as CIM-BIOSYS (Weston *et al.* 1989, 1990, 1991).

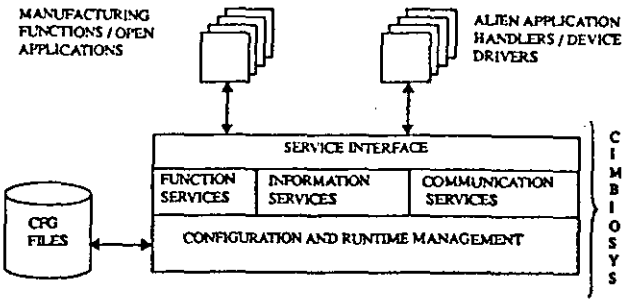


Figure 1. A functional view of CIM-BIOSYS.

This notion has been supported by the requirements specified within the CIM-OSA reference architecture (CIM-OSA 89). CIM-BIOSYS consists of a number of functional blocks as shown in Figure 1.

The manufacturing functions and applications shown in Figure 1 are in this context viewed as being those processes which perform some part of a distributed and yet integrated manufacturing operation. Typical examples include a cell controller or a scheduling application. The device drivers hide or cater for the diversity of both functionality and implementation of system resources. Typical examples of system resources include shop floor machines, proprietary databases and human operators.

The need for an integration infrastructure, such as CIM-BIOSYS, is now widely accepted by the manufacturing systems integration research community. By enabling the creation of 'open' software, the use of an integration infrastructure can promote the more systematic generation and change of flexibly integrated systems. This provides a means of dealing with the high levels of complexity found in most manufacturing organizations (Weston *et al.* 1989, 1990, 1991).

4. A proposed model for fully integrated machine vision

When using an integration infrastructure, such as CIM-BIOSYS, to integrate practical manufacturing systems, three types of modular building block of CIM systems can be identified, as shown in Figure 2. The following summarizes the main features of each module type, while a more detailed treatment of a proposed breakdown of each module type is presented within the section covering the proof of concept implementation.

4.1. *Type 1 modules: point process functionality modules*

In this work the term 'point process' is used to describe a single manufacturing process, in this case a machine vision inspection system.

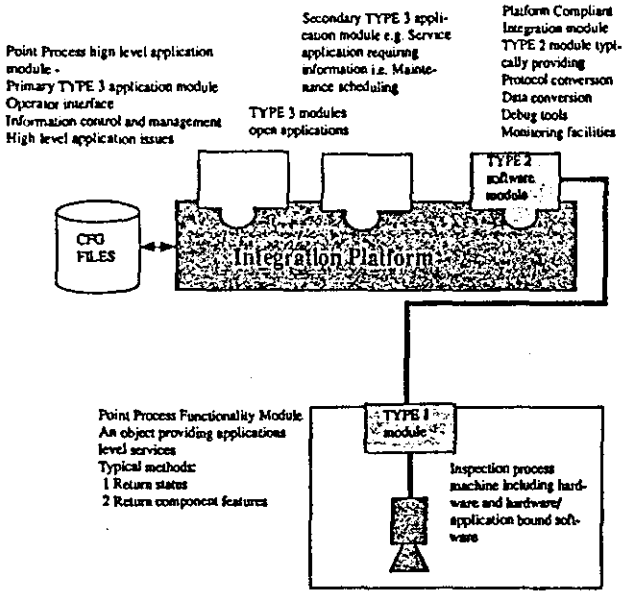


Figure 2. A representation of an integrated vision machine.

Type 1 machine vision inspection modules embrace the hardware providing the point process functionality, specifically the image capture card and associated processing hardware, plus the software providing the application and hardware bound functionality. This includes functionality that is required to interface to hardware components, primarily the driver software for the vision chips on the image-grabbing hardware, or software that is required to operate within speed constraints, such as control software or vision processing software.

This module forms a point process object which provides a level of services above that provided by the integration services within CIM-BIOSYS. These point process services could embrace such functionality as 'inspect PCB', 'accept PCB AB123 inspection criteria' and 'return features of a component of this model type' as might be used during the normal operation of the process, or 'return self-diagnostic check information' a service typically available to more general applications running on the integration platform.

4.2. *Type 2 modules: platform compliant integration modules*

This module should be completely invisible to the open applications (or type 3 modules) which make use of the application services provided by the type 1 point process object module. It mainly comprises protocol conversion facilities. permitting CIM-BIOSYS compliant applications to communicate with the remote point process, fielding messages from CIM-BIOSYS, converting them such that they can be interpreted by the type 1 process module and vice versa.

Additional facilities are incorporated in type 2 modules for integration debugging as this key functionality linking the application software functionality to the remote point process object is essentially the 'communications driver' of old, a software domain fraught with problems. Engineering access is provided through a display system typically enabling interaction both up to type 3 modules and down to type 1 modules to be isolated and proved.

4.3. *Type 3 modules: open applications*

Arguably this type of module could be used to implement all functionality associated with a point process application that can be lifted out of the traditional remote standalone application. Through achieving such a separation the application's functional software can be moved into an environment which today offers very sophisticated tools for software generation and debugging. Here great benefit can be derived from a marriage of advanced manufacturing technology and the rapid developments emerging in the field of computer science (i.e, in terms of hardware, software and information technology). This, together with an implementation approach based on the use of an integration infrastructure, can provide an environment where the key requirements of ease of integration, flexibility and ease of change of a manufacturing process can flourish.

Type 3 modules will typically embrace such functional capabilities as information management, operator interface, off-line analysis and non-time-critical application functionality.

Type 3 modules can be created so that they comprise a fully compliant integration platform point process (or open application) and as such can be complemented by further modules of the same status as indicated in Figure 2, which make use of the services provided through the other modules of the model. Complementary applications could typically include maintenance scheduling or process monitoring, the sort of task which in the past has been considered to be non-essential but is now a very real necessity when tuning a manufacturing facility to today's high-efficiency requirements.

Open applications have access to the platform of services within the integration infrastructure. It is this situation that allows the building of an information support system for the application from various information sources available through the infrastructure.

This paper describes the use of information retrieval via both relational technology and the object model. A proof of concept implementation is described which inspects alien components, extracts their features and searches various information sources for component objects with matching feature attributes in order to

classify the component. Following successful classification a view of the specific component model information is used to derive more detailed features for use during inspection. This task of information management and manipulation is termed information view provision (IVP). It is implemented in the form of complementary open application software which runs on CIM-BIOSYS, making use of available information services and resources. IVP is covered in more detail in Section 8 of this paper.

5. A proof of concept implementation; an integrated application enabling the automated vision inspection of surface mount components

Figure 3 shows an overview of the 'proof of concept' system, which was constructed to illustrate how enhanced operational performance can be realized when inspecting PCBs and components placed on or through them. Figure 3 illustrates the use of three module types to implement both the vision inspection application and the supporting IVP application. In the case of the IVP application, use was also made of application level information services provided by both an INGRES database and a GENERIS knowledge base.

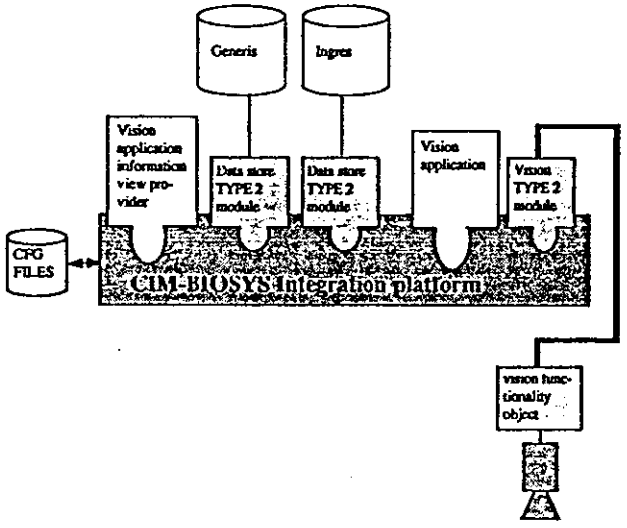


Figure 3. Overview of elements making up the proof-of-concept system.

5.1. *The point process functionality module: an information driven, object-oriented vision inspection system*

Figure 4 shows the basic architecture of the type 1 module, which embodies the notion of a separation of

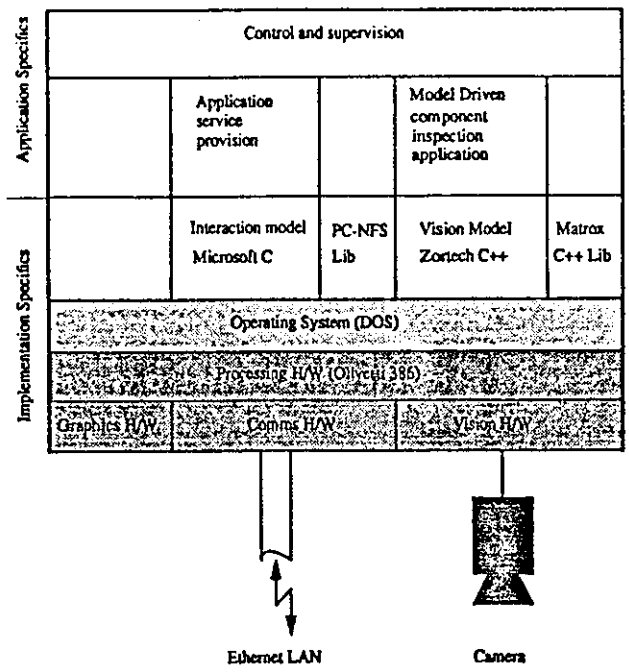


Figure 4. Elements within the point process functionality module: H/W hardware; LAN, local area network.

implementation-constrained technology issues from application level issues. This separation allows a replacement, development or upgrading of the technology without the necessity to replace the application code which may represent a major user investment, that is the application requirement such as 'inspect PCB' still remains the same, while the achievement of the goal may be improved by the use of improved implementation technology. The simple two-level decomposition of vision-technology-constrained issues and application issues is in fact subject to greater decomposition and the point at which the technology ceases to influence the application may vary.

5.1.1. *A low-level vision-processing model.* To determine the required functional decomposition of a vision machine as a manufacturing process the vision operation was first modelled. The philosophy of trying to move all application considerations up a hierarchical stack to increase the scope of the derived subobjects was applied.

The analysis methodology used was that of Coad and Yourdon (1990) based on the principals of classification structure (or GENeralization– SPECialization structure) and assembly structure (or whole–part structure). The resulting essential model of grey-level and binary image pre-processing including facilities for segmentation combined the simple assembly of a single whole image and multiple window images with multiple classifications of derived or transposed images (i.e. edge enhanced,

thinned, etc.). During the design phase the model suggested by Booch (1991) was derived, which embraces his notion of object evolution. Also from the design model an implementation-constrained model was developed for the hardware available, in this case a Matrox PC-AT vision board and associated driving software. The model was implemented in Zortech C++ to retain the object-oriented shape of the model and to enable and test the claimed benefits of object-oriented code (e.g. code re-use, and code redevelopment through encapsulation and inheritance). Figure 5 shows the different stages of the vision technology model.

By the same process, feature extraction was modelled by the creation of point-set objects, arcs, boundaries and networks, with associated features, for example, length, perimeter, area, and number of nodes and arcs. The features extracted and the methods created to generate them were designed around three parallel applications, including SMT component inspection (as described in the following section), bare PCB inspection and character inspection for integrated-circuit identification. A number of application domains were studied to identify generic requirements. This led to the definition

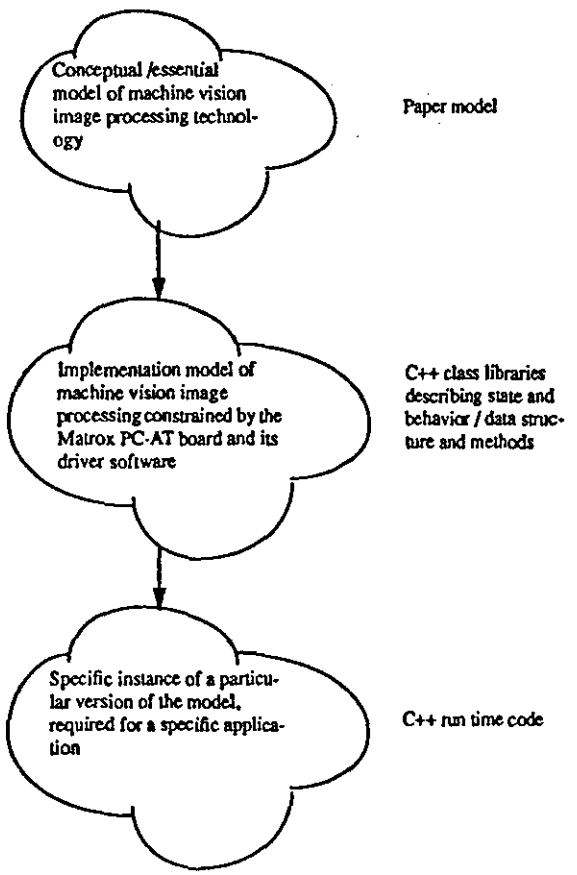


Figure 5. Three-stage vision technology model.

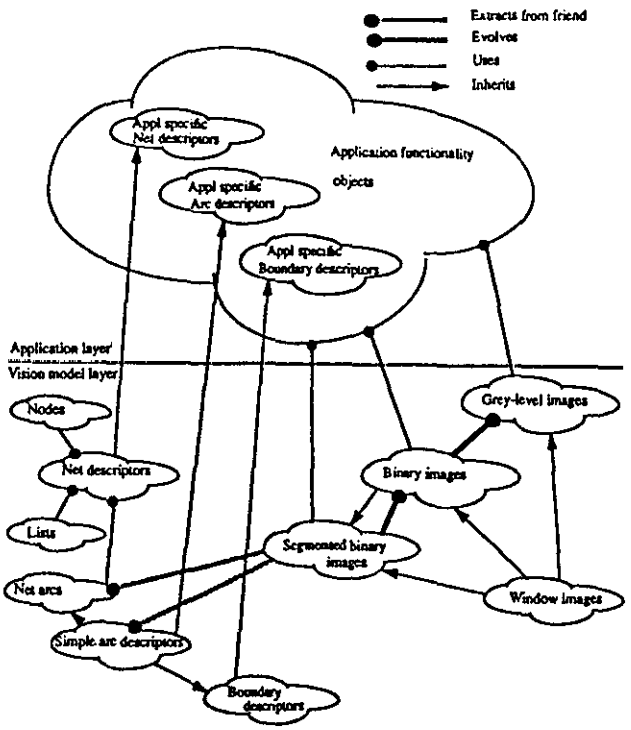


Figure 6. Vision-processing model using the adapted Booch (1991) class diagram notation.

of class hierarchies of a generic nature; again they contain no constraining application specific-code.

Figure 6 shows the model and its relationship with applications. The notation used to describe the model is that of Booch (1991) class diagrams, with the addition of an 'evolution' and an 'extracts from friend' relationship, required to describe fully the nature of the technology.

Further details of the low level vision model, which are not appropriate in the context of this paper, are available from the authors.

5.1.2. *The inspection application: information-driven component inspection*

As shown in Figures 4 and 6, the implemented application code sits above an implementation of the vision model, instantiating the required vision objects (which typically will be representations of either complete images or window subsections of them, as whole-image and window-image objects) and firing messages at them to achieve vision processing transformations to enable segmentation. Application-specific classes, used to describe live objects within a scene, inherit state and behaviour from arc, boundary and net classes as appropriate while adding the required methods to control the

behaviour of their specific objects, for example a 'component leg' class inherits from a 'simple arc' class, a 'PCB net' class inherits from a 'net' class. Arc, boundary and net-based objects are then instantiated to describe the state and behaviour of the required live features.

The application is designed using the same analysis principles as described earlier using a model of a generic surface mount component and is implemented in Zortech C++. Very briefly the application seeks to derive benefit from the fact that the complete point process (the complete vision machine) is integrated within a system, thereby offering services such that the provision of information is eased. The application extracts basic features from whatever component is placed at whatever position in its field of view. It then classifies the component as being one within a pre-defined class, for example SO8, SO16 or SOT. The system can rely on a restricted local set of component descriptions loaded prior to inspection, restricting the system to information local to the application (as typically could be down-loaded to a component placement machine with incorporated component inspection facilities). Alternatively it could search via the integration platform information services for a remote component description to match the live features extracted from the component under inspection. The system then returns to the original image with specific information as to the position of the pertinent features of the classified component derived from a view of the component model (future implementations could include additional information as to the best way to process those features). With this information a more accurate description of features can be generated while retaining flexibility. The information is derived from a product model instantiated within a data base within the integration infrastructure, such that any component that can be described using the model is capable of being inspected (providing inherent flexibility).

5.1.3. *Interaction issues.* Interaction is required between the PC-based vision machine comprising the type 1 point process and the platform compliant integration module which resides on a Sun network. The hardware chosen to implement this link was an ethernet communications board mounted in the PC and linked into the Sun network. PC-NFS, a software package providing extensions to DOS-based C, was used to permit the use of Unix-socket-based communications. As shown in Figure 4, this system provided the implementation constraints for the type 1 module interaction model. The model comprises the message-building and stripping functions.

The messaging application in Figure 4 comprised the handling dialogue of the messages providing the services of the type 1 module these typically include the following STATUS request, START vision process, REQUEST component features (live model) and accept component design models (of components to be inspected).

In the proof-of-concept application, control and supervision (see Figure 4) comprised the initialization and setup functions, most of the control being done through service requests from the supervising type 3 module running on the integration infrastructure.

5.2. *The type 2 platform compliant integration module*

The essential functional requirement of this module is one of protocol conversion and, as such, the requirements of this module could be common to any manufacturing point process. Figure 7 gives details of the elements within the module.

The application section of this module contains all the functionality specific to converting the CIM-BIOSYS compliant messages and the data contained within them to a form required by the system on which the type 1 process is running, in this case an Olivetti 386 running DOS.

Experience has shown that extra functionality must be made available within type 2 modules to enable the overall architecture embracing the three types of module to be partitioned and tested in isolation, type 2 to type 1 to prove and monitor communications from the Sun network to the remote machine and from type 2 to type 3 to prove successful communication across CIM-BIOSYS.

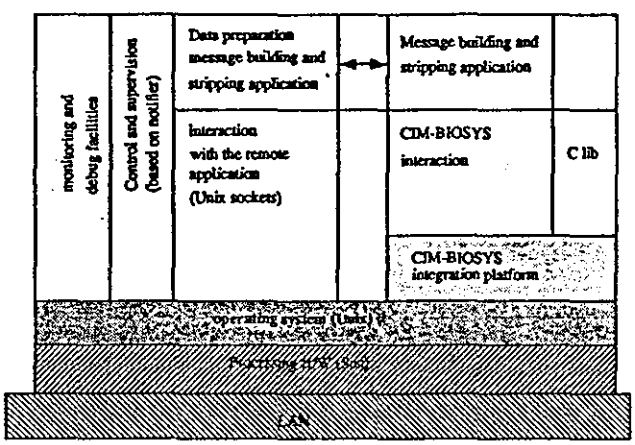


Figure 7. Elements within the platform compliant integration module (driver): H/W, hardware; LAN, local area network.

5.3. *Type 3 open application modules*

In the proof-of-concept system the type 3 application modules provide overall supervisory control, high-level

application issues, support information management and display of live component features. Being CIM-BIOSYS compliant, this module can be viewed as an open application sending and receiving messages to and from the platform of integration services. A support IVP application also runs as an open application on the platform (this being described in detail in the next section). These two type 3 application modules send and receive messages to and from each other and to and from the type 2 platform-compliant integration module (see Figure 3 detailing the elements making up the system). As previously mentioned, other type 3 open applications may also access the functionality of the type 1 point process module by addressing its type 2 platform compliant integration module.

Future enabling technology could permit much of the application functionality to be achieved as type 3 modules. In the proof-of-concept system, this is assumed; so the live features are passed up to this level for use in some inspection function, thus leaving the component-model-driven type 1 module free from inspection application detail.

This completes the overview of the proof-of-concept vision inspection system. The following describes the information support mechanisms implemented through the CIM-BIOSYS integration infrastructure.

6. Information support via application services

This section describes the form of IVP applications and the required database drivers which provide the application level services via the CIM-BIOSYS integration infrastructure.

As mentioned previously, the type 3 vision inspection application provided information management and, as such, initiates the search for component objects. An information request is sent to the view provision application specifying the component features, in this case the area and perimeter. The vision application expects to receive a reply from the view provider either saying such a component does not exist or confirming its existence and providing information derived from the component model such that the vision application can extract detailed features from the inspection scene. In this application, components are stored in both the relational and the object model form implemented using an INGRES database and a GENERIS knowledge base. The schema used for both systems is based on the model given in the report of the Electronic Data Interchange Format (EDIF) Printed-Circuit Board Technical Subcommittee (1990), as described in the following section.

6.1. *The Electronic Data Interchange Format printed-circuit board model*

The information requirements of the inspection process are very specific, as are the requirements of the many other processes within electronics manufacture. In order to support these processes with the required manufacturing information, each process requires a different 'view' of a global data store which holds a description of the as-designed product. In PCB manufacture this design is typically produced using a Computer-aided design package. A wide range of such packages exist and each produces (or holds internally) a description of the designed PCB in its own proprietary format. Not only are the formats different but their information content differs also. For example, one format may hold information describing the shape of a component leg whereas another may not. An internationally accepted PCB representation would ease the problems of the design to manufacture interface and obviate the need for multiple interfaces to cater for proprietary formats.

The EDIF community has been addressing this problem over the past 2 years by producing PCB extensions to the accepted EDIF interchange standard for very-large-scale integration designs (EDIF version 2·0·0). The first step in producing these extensions was to model conceptually the PCB in a modified version of IDEF-1X (Bravoco and Yadar 1985, Electronic Data Interchange Format Printed-Circuit Board Technical Subcommittee 1990). This was deemed necessary to resolve fully the complexity of the entities (together with their attributes and relationships) which constitute a PCB. This model was then used as a reference to facilitate the production of new EDIF syntax for PCB representation. In addition to the graphical IDEF-IX model, a corresponding model was produced in the information modelling language EXPRESS, which is the modelling language adopted by the STEP (Standard for Exchange of Product data) initiative (International Standards Organization 1990, 1992).

Although EXPRESS was not designed to be a database definition language, it provides a suitable computer-readable information model which can be used (in whole or in part) to structure databases. Within the SI Group, the EDIF EXPRESS information model has been used to produce both relational and knowledge base schema to provide data stores for the CIM-BIOSYS platform. The vision inspection application demands and receives the information necessary to perform its inspection from these data stores via the CIM-BIOSYS platform on request from the view provision applications as described in Sections 8 and 9 below. The vision inspection application asks for information regarding the perimeter and area of a particular surface mount device. The relevant

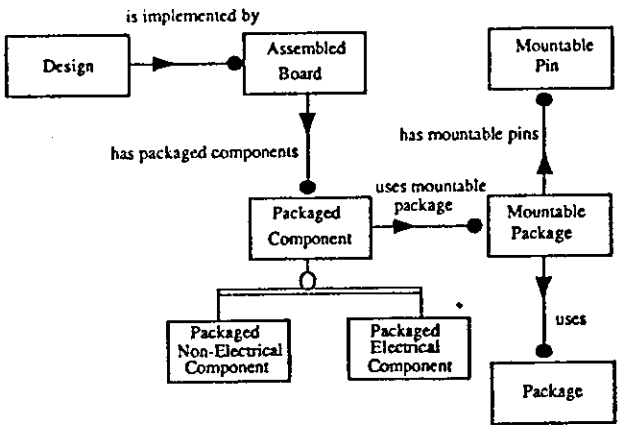


Figure 8. Part of the EDIF model for PCB, in IDEF-1X notation.

part of the PCB model which corresponds to component shape and size is that containing package information. This part of the model with its inherent hierarchy is shown in IDEF-1X and EXPRESS form in Figures 8 and 9 respectively.

6.2. *The relational view provision application*

A detailed description of the principle of CIM-BIOSYS (SI Group 1991) would identify a requirement to separate knowledge of integration issues, which is normally embedded within an application, from the functionality of the application and to store the integration knowledge within CIM-BIOSYS system tables. In order to accomplish this within the information provision services a concept known IVP has evolved and has been implemented within the CIM-BIOSYS platform. The basic philosophy behind this has been to hide the particular knowledge about a data store, typically which database, which machine the database is on, and the various tables and attributes contained within a database, thus allowing the application to reference every data item or set of data items as a system-defined object name. The IVP then maps these system data into the relevant database calls and executes these statements on the host database, finally passing the information requested by the application back to the application in some predetermined format.

In order to create a relational view provider (RVP) a three-Schema methodology (first suggested by the American National Standards Institute Standards Planning and Requirements Committee (ANSI-SPARC) Database Management Systems Study Group (Tsichritzis *et al.* 1978)) was adopted. This consists of three views: the external view, the conceptual view and the internal view (Figure 10).

```
ENTITY design;
is_implemented_by : assembled_board;
identification : INTERNAL ID_STAMP;
END_ENTITY;

ENTITY assembled_board;
is_based_on : bare_board;
identification : INTERNAL ID_STAMP;
has_packaged_components : SET [0:#] OF packaged_component;
END_ENTITY;

ENTITY packaged_component
SUBTYPE OF (component)
SUPERTYPE OF (ONEOF
(packaged_electrical_component, packaged_non_electrical_component));
uses_mountable_package : mountable_package;
component_type : INTERNAL PACKAGED_COMPONENT_TYPE;
END_ENTITY;

ENTITY mountable_package;
has_mountable_pins : SET [0:#] OF mountable_pin;
mounting_height : INTERNAL REAL;
lead_form : INTERNAL STRING;
package_class : INTERNAL STRING;
body_volume : OPTIONAL INTERNAL three_d_spec;
occupied_space : OPTIONAL INTERNAL three_d_spec;
true_shape : OPTIONAL INTERNAL SET [1:#] OF three_d_spec;
ses_package : package;
END_ENTITY;

ENTITY package;
has_pins : SET [0:#] OF pin;
lead_form : INTERNAL STRING;
package_class : INTERNAL STRING;
body_volume : INTERNAL three_d_spec;
occupied_space : INTERNAL three_d_spec;
true_shape : OPTIONAL INTERNAL SET [1:#] OF three_d_spec;
END_ENTITY;

ENTITY mountable_pin;
position : OPTIONAL INTERNAL point;
side : INTERNAL pin_side;
END_ENTITY;

ENTITY three_d_spec;
figure : INTERNAL polygon;
height : INTERNAL REAL;
END_ENTITY;

ENTITY polygon
SUBTYPE OF (closed_geometry);
edge_list : INTERNAL LIST [3 : #] OF line_segment;
END_ENTITY;

ENTITY line_segment;
start_point : INTERNAL point;
end_point : INTERNAL point;
END_ENTITY;

ENTITY point;
x : INTERNAL REAL;
y : INTERNAL REAL;
END_ENTITY;
```

Figure 9. Express listing for mountable package entities.

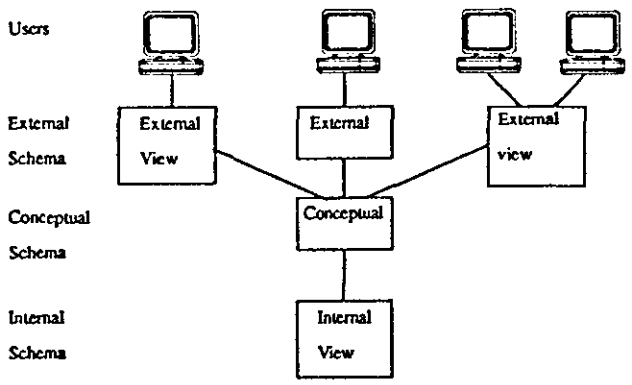


Figure 10. Three-schema methodology for database management.

Each of these views will be described in more detail in the following sections.

6.2.1. *The external view.* The basic premise behind this view is the fact that the user of the system, whether it be an application or a human, does not need to have any knowledge about the actual database structure; they request data by referencing object names.

In order to accomplish this within CIM-BIOSYS, the system administrator or the system designer is given the ability to set up object names which are mapped onto attributes within the particular databases, for example AREA which is an object name that is mapped onto the attributes area of package within a particular table. The important facts to note are the lack of knowledge needed about the physical database, and the immunity that this gives the application to change of the physical database; all that needs to be changed are the mappings—hence change time is drastically shortened and hence system response time to change is enhanced.

6.2.2. *The conceptual view.* This is an abstract definition of the database and represents the data and the relationships between the data without considering the physical resources available or the system that it is to be stored on.

The EXPRESS language is used as the conceptual definition language for the RVP; this is mainly because EXPRESS is computer readable and lends itself well to the definition of entities their attributes and their relationships with one another. The SI group have produced an EXPRESS to SQL (ISO 87) compiler which takes as input the EXPRESS model and produces amongst other items a set of SQL statements which describe the table structure for the implementation of the EXPRESS model. The EXPRESS model is then used by the RVP as the basis for producing the relevant SQL queries that will be needed to access the relevant data as specified by the requested data object.

6.2.3. *The internal view.* This is the lowest level and deals with the physical representation of the data and its organization.

Within the CIM-BIOSYS platform, a number of relational databases are able to be accessed using platform-compliant drivers implementing similar functional requirements to the level 2 platform-compliant integration module within the vision model. Typical databases include INGRES and ORACLE. The structured methodology used in creating these drivers is such that the creation of drivers for other relational databases can be achieved with ease, while the RVP can be used for any relational database because the SQL statements produced are completely non-proprietary using a small subset of the SQL statements as laid down by the International Standards Organization (1987), this subset being those statements that are regularly implemented in a consistent manner.

The SQL statements produced by the RVP are similar to the following set of SQL statements which is a request for the occupied space of a package (these are included for comparison with the requests made on the knowledge base):

```
create view v1 (line_segment_id,vl_x,vl_y)
as select line_segment.line_segment_id,point.point_x,point.point_y
from line_segment,line_segment_start_point,point
where line_segment.line_segment_id=line_segment_start_point.line_segment_id and
line_segment_start_point.point_id=point.point_id

create view v2 (line_segment_id,v2_x,v2_y)
as select line_segment.line_segment_id,point.point_x,point.point_y
from line_segment,line_segment_end_point,point
where line_segment,line_segment_id=line_segment_end_point.line_segment_id and
line_segment_end_point.point_id=point.point_id

create view v3 (line_segment_id,vl_x,vl_y,v2_x,v2_y)
as select vl.line_segment_id,vl.vl_x,vl.vl_y,v2.v2_x,v2.v2_y
from v1,v2
where v1.line_segment_id=v2.line_segment_id
```

```
create v4 (polygon_id,vl_x,vl_y,v2_x,v2_y)
as select polygon.polygon_id,v3.v1_x,v3.v1_y,v3.v2_x,v3.v2_y
from polygon,polygon_edge_list,v3
where polygon.polygon_id=polygon_edge_list.polygon_id and
polygon_edge_list.line_segment_id=v3.line_segment_id
```

6.3. *The knowledge-based view provision application*

In order to investigate the mechanisms required and the facilities offered by different data storage methods, the SI Group has been using the object model in the form of the GENERIS knowledge base (Instrumatic Da · Systems 1990). A knowledge base implementation of EDIF EXPRESS model for PCB has been created and used through an operator interface to serve the vision inspection application with the required information necessary to complete its task. The following section details the requirements for the 'knowledge-based view provider (KBVP)' based on the present semi-automatic implementation for GENERIS information retrieval. Details of the mapping from the EXPRESS PCB model to the GENERIS schema have been published previously (Murgatroyd *et al.* 1991).

6.3.1. *Information retrieval.* The vision application requests information regarding packages (components) with specific area and perimeter values. In response to this request, the view provider makes the necessary knowledge base queries to determine whether a corresponding package is present in the data store. If a corresponding package exists, the view provider makes further queries to extract the positions of the package pins so that the vision inspection application can continue its task. Reference to the EXPRESS listing in Figure 9 shows that a mountable package entity has an 'occupied space' attribute which consists of a collection of joined edges. The view provider must extract the x and y positions of these edges in order to calculate the area and perimeter of each package. The GENERIS query to list all the mountable packages in the database is as follows:

```
DISPLAY mountable_package of design
```

This query retrieves all the mountable packages for one particular PCB design. On the assumption that one of these mountable packages is called SO8, the subsequent queries to extract the x and y coordinates of this package are as follows:

```
FETCH x of start_point point of occupied_space space of SO8
FETCH x of end_point point of occupied_space space of SO8
FETCH y of start_point point of occupied_space space of SO8
FETCH y of end_point point of occupied_space space of SO8
```

These queries are issued from a GENERIS command file and their values manipulated to determine the area and perimeter of the package. If these values correspond to those sent by the inspection application, further queries can be made to extract the positions of the pins within the package. It is these pin positions which form the basis of the information required to support the vision application:

```
FETCH x of point of mountable_ pin of SO8
FETCH y of point of mountable_pin of SO8
```

It should be noted what the queries are constructed from the corresponding entity and attribute relationship names in the EXPRESS listing in Figure 9. The entity names used in these queries are 'point', 'mountable_package' and 'mountable_pin ('space' was set up as an alias for 'three_d_spec' within the GENERIS environment). Attribute relationship names used in the queries are 'x', 'y', 'start_point', 'end_point' and 'occupied_space'. These attribute relationship names are necessary when an ambiguity arises in a query using the entity names, for example the query

```
FETCH x of space of SO8
```

would give rise to ambiguities since x could refer to the 'start_point' or 'end_point' (both of entity type point) and shape could refer to 'body_volume', 'true_shape' or 'occupied_shape' (all of entity type 'shape', alias 'three_d_spec').

The view provision application is then greatly simplified as the knowledge associated with the EDIF model resides within the knowledge base, removing the necessity for a conceptual schema and internal schema section within the KBVP. It must be stressed, however, that the generic nature of the RVP provides the conceptual and internal schema requirements for most relational databases accessed via SQL. Thus in both cases the application aspects of the view provider make use of information retrieved at the external schema level and provides the application-specific functions required to manage the information search and to manipulate the

retrieved information, providing the view required by the vision application.

7. Conclusions

Before benefit can be gained from information-driven processes based on models similar to that proposed in this paper it is essential that the concepts of flexible integration be embraced, together with the provision of a suitable integration infrastructure. CIM system installation of any significant scope must not be attempted without an integration infrastructure and platform of services in the same way as software applications running on a computer would not be used without an operating system. The following headings address the requirements for future processes as proposed in the introduction.

7.1. *Increased ease of integration*

The use of an integration infrastructure such as CIM-BIOSYS provides integration level services for application interaction and information provision; this paper has proposed an application decomposition made possible by the use of such an infrastructure. The proposed decomposition provides application level services from its type 1 point process functionality module which are used by higher-level applications running on the platform. The intermediate type 2 driver provides the single point of access to the point process functionality. Multiple applications on the platform can access the type 1 functionality, making use of application level services; typical applications include management information, maintenance information and process optimization. The integration of these applications with the type I functionality is done through the use of messaging to the platform and the appropriate entry of data in the platform configuration tables, easing the complexity problems associated with application integration. This same philosophy applies to the interaction integration between the main application and the view provision application and the view provision application and the data stores.

The use of CASE (computer aided software engineering) tools to encapsulate methodologies for systems analysis, design, implementation and reverse engineering aid the systems developer for both primary implementation of software and rework. The development of CASE tools in the area of integration and platform application building is a current theme within the SI Group at LUT; this work will improve facilities for the use of such integration infrastructures and encourage vendor provision of compliant applications.

7.2. *Increased ease of change*

At an integration level, ease of change is embraced by the configurable nature of CIM-BIOSYS. Integration issues such as where data stores reside and on what machine the high-level applications are running are controlled via the platform configuration tables. These tables can be changed at system administrator level through the use of graphical user interface and platform support tools.

At the vision machine functionality level the decomposition of domain-specific technology issues and application issues yields the separation of the vision model and its constrained implementation from the application code. A required change in the implementation technology can be eased by the replacement of the implementation constrained model while the application code is retained. The use of object-oriented techniques were useful in impletmenting this decomposition, with the application classes inheriting the domain-generic extracted feature classes. CASE tools based on the domain-generic model of vision (at present under investigation at LUT) will increase the ease with which change can be enabled.

The benefits of object orientation were also evident in enabling the production of code which retains the shape of the conceptual model of a technology. Also of great importance is the ability to modify the application in line with any change to the product model. If an application is based on extracting features from a component object, and a new attribute is required for a component model during its life cycle, the object-oriented features of inheritance and encapsulation permit a new model to inherit all the old attributes (or feature extraction methods) and to add the new requirements without any knowledge of the detail of the original application.

7.3. *Increased flexibility*

The use of information models to drive the application ensure the flexible nature of the information provision application in that they can respond to new components within a database or knowledge base so long as the components adhere to the model (in this case EDIF). A similar situation applies to the level 1 point process functionality module, in that it will respond to any component so long as it complies with the model on which the application is based.

In general, great flexibility is achieved through the availability and use of platform services and applications providing higher-level services running on the platform, which can typically be used to structure applications in the manner of that described. For example component

classification is done via a 'view provider' which searches the platform via the information services for a component object with the captured features.

7.4. *Summary*

It is the features of increased flexibility, ease of change and ease of integration embodied in the structure of the proposed model that will enable the generation of future vision machines that can adapt to changing requirements and (as hinted at in the implementation and information view provision described) lead to intelligent reconfiguration or information self-supporting applications.

7.5. *Future developments*

As a final point it is important to note that the proposed three-module system represents an architecture which enables the integration of systems during a migration period from present to future implementations of CIM-BIOSYS. At present, the CIM-BIOSYS integration platform is available to offer integration services to systems running under BSD 4·2 UNIX on Sun workstations, in the future the platform will be available on a number of processing systems (current work will enable its use in the near future on any system running UNIX System VII). Given the availability of CIM-BIOSYS on the processing system running the type 1 point process functionality module, parts of the type 1 module and the type 2 platform compliant integration module would become redundant and the type 1 and type 2 modules would be combined to enable the point process functionality module to become a CIM-BIOSYS compliant open application running directly on the integration infrastructure.

Acknowledgements

The authors would like to acknowledge the contribution made to the work described in the paper by all members of the SI Group at LUT, particularly Professor R. H. Weston who leads the group, Ian Coutts for his advice on writing applications for CIM-BIOSYS and Jak Gasgoigne whose original software underlies CIM-BIOSYS and its open applications.

References

BOOCH, G. B., 1991, *Object Oriented Design* (Benjamin-Cummings, New York). SI GROUP, 1991, The CIM-BIOSYS platform. *Internal Document* (SI Group, Manufacturing Department, Loughborough University of Technology).

BRAVOCO, R. R., and YADAV, S. B., 1985, IDEF, a methodology to model the information structure of an organization. *Journal of Systems and Software*, **5**, 59–71.

CIM-OSA, ESPRIT CONSORTIUM AMICE (eds), 1989, *Open System Architecture for CIM* (Springer, Berlin).

CAOD, P., and YOURDON, E., 1990, *Object-oriented Analysis* (Prentice-Hall, Englewood Cliffs, NJ).

ELECTRONIC DATA INTERCHANGE FORMAT PRINTED-CIRCUIT BOARD TECHNICAL SUBCOMMITTEE, 1990, Conceptual model of a PCB version 9 1990. *Report*.

INSTRUMATIC DATA SYSTEMS, GENERIS release 3.1. 1990 *Report* (Instrumatic Data Systems Ltd, Marlow, Bucks., UK).

INTERNATIONAL STANDARDS ORGANIZATION, 1987, Database language SQL. *ISO Standard No. TC97/SC21/WG3 N117.*

INTERNATIONAL STANDARDS ORGANIZATION, 1988, *ISO Standard No. TC184/SC4/WG1.*

INTERNATIONAL STANDARDIZATION ORGANIZATION, 1990, STEP Standard for the exchange of product model data. *ISO Standard No. TC184 SC4.*

INTERNATIONAL STANDARDIZATION ORGANIZATION, 1992, Product data representation and exchange—Part 11. The EXPRESS language reference manual. *ISO No. DIS 10303-11.*

MURGATROYD, I. S., WESTON, R. H., CLEMENTS, P. and COUTES, I. A., 1991, Knowledge base implementation of EDIF PCB conceptual model. In *Proceedings of the Conference on Electronic Data Interchange Format*, Montpeller, 1991.

SAKAKIBARA, M., and MATSUMOTO, K., 1991, Approach to CIM in NIPPONDENSO Kota Plant. *International Journal of Computer Integrated Manufacturing*, **4** (5), 279–287.

TSICHRITZIS D. C., and KLUG, A. 1978, The ANSI/X-3/SPARC DBMS Framework Report on the Study Group on Database Management Systems. *Information Systems*, 1, 173–191.

WESTON, R. H., HODGSON, A., COUTTS, I. A., MURGATROYD, I. S., and GASCOIGNE, J. D., 1989, Integration tools based on OSI Networks. In *AUTOFACT Conference Proceedings*, Dearborn, MI, 1989 (Society of Manufacturing Engineers, Dearborn, MI), pp. 17.1–17.14.

WESTON, R. H., HODGSON, A., COUTTS, I. A., MURGATROYD, I. S., and GASCOIGNE, J. D., 1990, Highly extendable CIM systems based on an integration platform, In *Proceedings of the Computer-Integrated Manufacturing Conference*, Gaithersburg, MA, May 1990 (National Institute of Standards and Technology, Washington, DC), pp. 80–94.

WESTON, R. H., ZHANG, P., MURGATROYD, I. S., COUTTS, I. A., and HODGSON, A., 1991, Soft integrated assembly systems. In *Proceedings of the Fourth World Conference on Robotics Research*, Pittsburgh, PA, 1991.

Machine vision and its integration with CIM systems in the electronics manufacturing industry

by John Edwards

Loughborough University of Technology

During recent years the electronics manufacturing industry has had to cope with very rapid developments in semiconductor technology, while also having to adapt to the changes in manufacturing philosophy which are being applied to most industry sectors. During this period machine vision has become an integral part of the automation being used to increase the efficiency or quality of the printed circuit board (PCB) fabrication and assembly processes which are required for this new miniature technology. This article outlines developments within the electronics manufacturing industry, examines the role of machine vision and the requirements for inspection during PCB assembly. The experiences of three major manufacturers are presented. The article concludes the need for the structured integration of vision machines within manufacturing systems to increase both flexibility and reliability.

Developments affecting the electronics industry

In most sectors of industry, market forces have dictated the need for changes in manufacturing philosophy. 'Just-In-Time' manufacturing methods have been adopted by many manufacturers, and even more widespread has been the need for significant increases in product quality so that companies can remain competitive. Computer-integrated manufacturing (CIM) although being embraced by many forward-thinking companies, has proved to be a more elusive goal. Program downloading from host to target systems and management information systems gathering local data for central planning and monitoring functions are common enough, but the optimised utilisation of information generated and used by the many and varied computer systems and programmable machines in use throughout today's manufacturing industry is still a long way off.

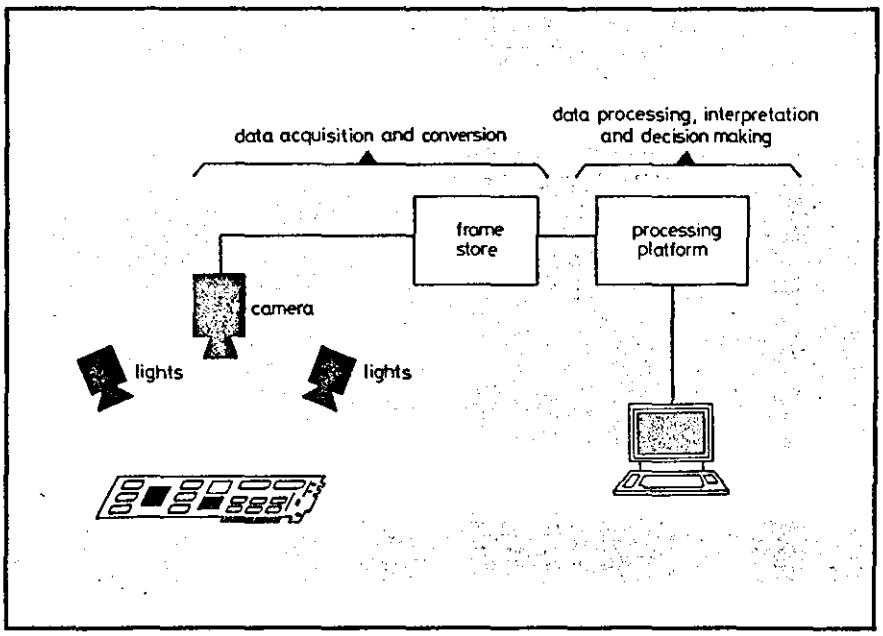


Fig. 1 The basic elements of a vision machine

The electronics manufacturing industry has been subject to change on a number of fronts. Both market pull and technology push have dramatically influenced such developments.

Market pull

The market demand for product variety and the problems of reduced product lifetime can lead to manufacturing problems, such as increased downtime due to product changeover and increased work in progress, unless modern technology can be correctly applied to alleviate such problems. Furthermore, time-to-market has become critical and must be reduced for a company with high levels of functionality and quality in its new products to gain an early foothold in the market place.

Technology push

Developments in semiconductor technology are providing increased functionality p.u. area at the PCB level, offering opportunities for increased performance and miniaturisation. This is very attractive to the electronic product designer, but is causing many PCB assembly and test problems. Surface mount technology (SMT) has been established; fine pitch technology (FPT) and chip on board (COB) are already being used. This has led to much tighter assembly and inspection tolerances being required. It is within this environment that machine vision, applied to PCB manufacturing equipment, is beginning to play

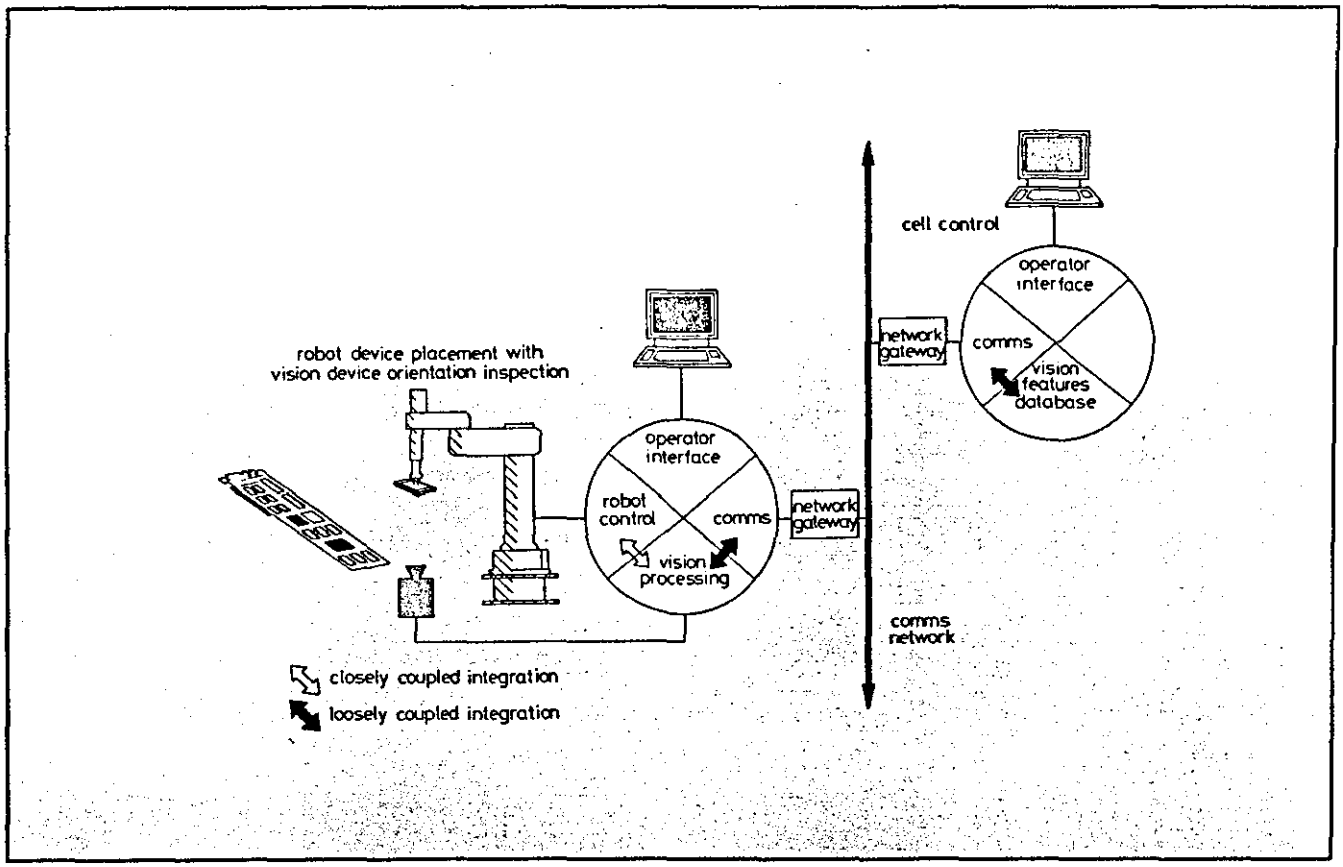


Fig. 2. Examples of two classes of integration: closely coupled and loosely coupled

a very important role, offering methods of flexibly automating inspection and assembly processes.

The increasing cost and risk associated with the complex automated processes now required to assemble this miniature technology are causing the structure of the electronics manufacturing industry to change. Often small entrepreneurial design companies are no longer considering manufacture, but instead favour contracting out their work to large subcontract operations who specialise in SMT assembly. This technology push is also providing the enabling technology to help solve some of the manufacturing problems (including the introduction of CIM), providing high-speed data-processing facilities. These technology advances are particularly important to the successful use of machine vision.

CIM concepts and potential benefits

The increasing use of computer tools to assist in the design of products, organisation of manufacturing functions and automation of shopfloor processes has led to a realisation that information is a valuable resource. CIM concepts are evolving to more effectively utilise information [1], and hence improve productivity and efficiency levels.

Potentially enormous benefit can be gained from CIM; put simply, the reuse of information. Information is entered into computer systems and programmable machines in numerous different locations, used for one purpose and stored or deleted. Valuable design office information laboriously entered through keyboard and mouse is held on design office CAD systems storage media. Equally valuable process information is acquired continuously by shopfloor machine sensors and often used only for local control and monitoring. A camera can operate as such a sensor, functioning to collect raw data relating to an image or scene for subsequent analysis by the processing sections of a vision machine.

The essential elements of a vision machine

Vision machines should be classed as one type of sensor. Generally, they comprise (Fig. 1)

- lighting equipment to control the illumination of the scene to be viewed;
- a sensor, usually a camera; either Vidicon line scan solid-state or more often a two-dimensional solid-state camera, consisting of a scanned array of light-sensitive silicon, with processing hardware to convert the scanned output to an analogue video signal format;
- a frame store consisting of a memory array, which will input the analogue video signal and convert it to digital format to capture a multi grey level description of the viewed scene;
- a data-processing platform to perform computations on the stored information, extracting meaningful information from the scene and interpreting this information.

Recent technology advances have allowed the component elements of vision machines to become more sophisticated. For example, simple frame stores, storing raw images in digital form, have evolved to vision engines, typically incorporating dedicated vision-processing hardware, multiple frame capture with colour capability and array multiprocessing. Increased sophistication of available elements has contributed to vision becoming a more usable tool within manufacturing. In particular, much improved image-processing facilities can lead to reduced cycle times and an ability to extract meaningful information from more complex scenes.

Two classes of integration

When automating electronic product manufacturing processes, vision machines are already a viable tool in application areas of both monitoring and control. In control applications where the vision machine must interface with some form of actuator system (typically a robot placement arrangement or an onsertion machine), computer integration can be used with significant benefit. Essentially there are two cases for integration (Fig. 2):

- local integration; where closely coupled programmable machines must communicate information between each

other, and the primary consideration is speed, e.g. the vision system and the actuator system;

- factory integration; where more loosely coupled programmable machines and computer systems are involved in the utilisation or generation of centrally created or utilised information.

At present, and in many cases justifiably due to speed considerations, the vision/actuator processing software is not structured such that integration to other manufacturing entities is easily achieved. Although less obvious, it is invariably the case that machine vision inspection also requires the vision machine to be closely coupled to other programmable entities, such as position actuators or lighting control. Inspection systems designed by different manufacturers form these interfaces in ways specific to their product or peculiar to a particular application.

Current application areas for vision machines in PCB manufacture and assembly

There are many current uses of vision machines in PCB manufacture and assembly. The following examples serve to illustrate this usage in the broad classifications:

- closely coupled feedback of information;
- loosely coupled feedback of information.

In both cases the information is used to improve manufacturers' efficiency and/or quality: the former to effect control of plant on a short time scale, and the second on a longer time scale, which in contemporary manufacturing systems always involves manual collection and interpretation of information.

Uses of vision with closely coupled feedback of information

☐ Position feedback is used for the realignment of the workpiece, following vision inspection of fiducial marks to provide information as to their exact position. Fiducial mark measurements are typically used in artwork alignment, drilling machine PCB alignment, onsertion/insertion machine PCB and component alignment, and robot pick-and-place component alignment. This feedback information can also be used in controlling solder paste screen printing machines during SMT PCB assembly, where fiducial marks on both the screen and the PCBs are viewed and compared so adjustments can be made to ensure the correct position of paste on component sites.

☐ Glue-dispensing control can be enhanced by using vision to inspect a set of demo glue drops prior to each PCB operation. The visual information is analysed and any necessary adjustments are made to the dispensing parameters. Fiducial correction is also incorporated on glue-dispensing machines. The Fuji GLII is a glue-dispensing machine which incorporates these features.

The use of vision with loosely coupled feedback of information

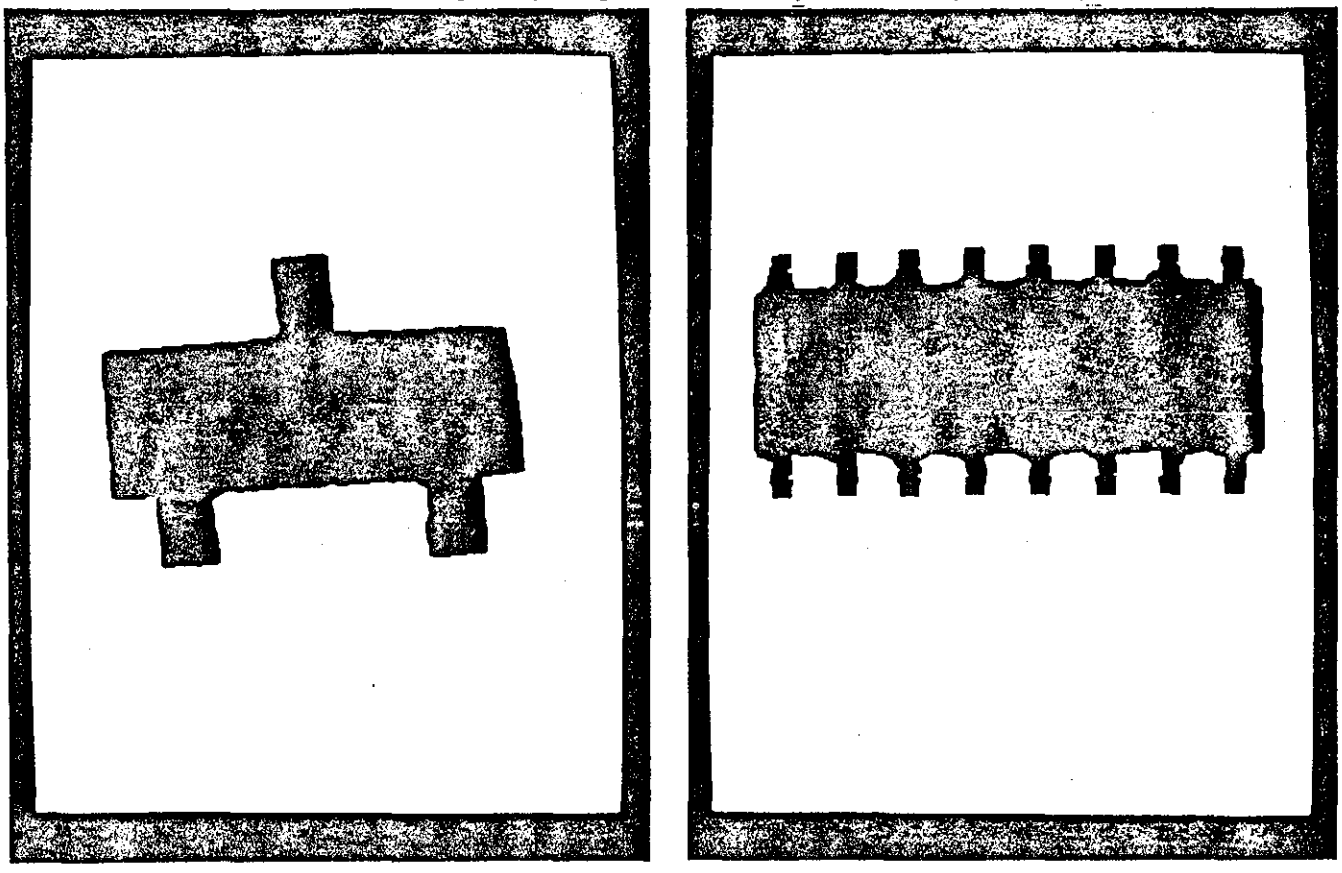
☐ Inspection during the manufacture of PCBs is achieved using vision machines to provide automated optical inspection (AOI) of artwork, innerlayers and finished boards.

☐ Inspection of loaded PCBs, both pre- and postsolder can be achieved using vision. Conventional though-hole technology boards use vision machines to identify leads through holes. The problems with inspection during SMT assembly are more complex, but commercial systems exist for the detection of component presence/absence, orientation and alignment.

☐ Vision machines are in use on drilling machines for assisting the generation of drilling information from PCB artwork, and checking and realignment of drilling information generated from CAD.

☐ Vision machines are available on rework stations to provide image enhancement to assist the operator. Final inspection stations monitoring solder joint integrity are also featuring image enhancement for operator assistance.

☐ X-ray vision machines are used in a number of areas: typically multilayer board inspection to check layer-to-layer misalignment and to assist in optimising the drilling of multilayer boards; component inspection to check for



a *b*

Fig. 3 Binary images of *a* SOT and *b* SOIC captured using backlighting on a vision system at Loughborough University.

defective bond wires and postsolder PCB inspection; checking solder integrity for voids and bridging, particularly under j-leaded quad packs where access to visible light vision machines is difficult [2]. In X-ray vision machines the grey level image describes levels of material thickness and density derived from the X-ray intensity measurements.

By looking in a little more detail at one of the successful implementations of vision machines within PCB assembly, some of the reasons for the success can be identified.

Realignment for component placement

A common application of vision machines is for alignment correction in order to accurately place SMT components. For FPT devices with a lead pitch of between, say, 0.38mm and 0.635mm the leads become too fragile to use mechanical centring with a pneumatic placement head. It is therefore necessary to use a vision machine to inspect the position of each device on the placement head after it has been picked up. Typically correction in X, Y is achieved by adjusting the PCB position, and theta correction is achieved by rotating the placement head. This system is usually implemented using a binary image capture technique thresholded to produce a silhouette of each device (Fig. 3). This captured image can be processed by the vision machine to verify correct lead count and pitch, and identify problems such as damaged components or incorrect component type (e.g. 14 pin SOIC). In practice, the benefits of using vacuum pickup and individual component inspection, with its additional checks and no mechanical centering, has meant it is being used for placing most types of small SMT packages, i.e. a level of programmable automation can be effected.

Alignment correction using fiducial mark position measurement (typically as used by Fuji Machine Mfg. Co., Zevatech, Dynapert and many other assembly equipment manufacturers) and automated optical inspection during PCB fabrication (typical manufacturers include Optrotech, AOI Systems, Lloyd Doyle) are further examples of the successful application of machine vision in the electronics manufacturing industry.

The success of machine vision in these applications can be attributed to an ability to assume basic axioms concerning the manufacturers' processes. For example:

- when generating offset information for a placement head, component position detection can be achieved by processing the binary image of the device outline.

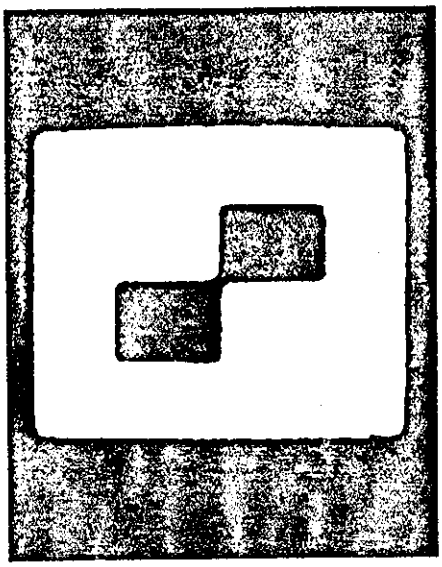


Fig. 4 Binary image of a fiducial mark captured using reflective lighting on a vision system at Loughborough University

This is commonly achieved by backlighting the device to produce a black object on a white background, thus producing a silhouette image to capture (Fig. 1). This means inconsistencies (or process variations), such as variations in device colour, surface finish or position and content of lettering on the device, can be ignored, as can the problems of extracting the relevant object from a confusing background. This enables the vision machine to have a consistent and fairly simple image to analyse. The required information for comparison with the analysed image are the characteristics of the ideal silhouette image of the component; typically size, lead count and lead pitch. These characteristics are known and stored in a library of all required components, making the training of the vision machine for new PCBs or at changeover a fairly simple task

- in fiducial mark correction the marks on the PCB are there only for the use of the vision system and are therefore designed specifically for easy detection and analysis (Fig. 4). The manufacturing process is adapted to ensure that fiducial marks are exactly as required by the vision machine and are not subject to process variation.
- in AOI systems, success can be attributed to the measurement task lending itself to consistent image capture and good definition of the object of interest against the background (Fig. 5), i.e. the tracks and pads against the PCB laminate or artwork film.

The success of these applications of vision can then be attributed to the ability to achieve the following:

☐ the design of the product to enable the effect of process variables to be minimised;
☐ the ability to simplify the vision task, leading to increases in operating speeds and more reliable operation through avoiding inconsistencies due to process variations;
☐ simplify the method of vision machine teaching, leading to a system that is both flexible and consistent, which may not require long installation times and/or product changeover times.

Having successfully implemented automated vision machines in some areas, the industry is looking for successful solutions to the very difficult problems of inspection during the assembly of SMT PCBs; typically solder paste inspection, checking position and paste volumes, presolder inspection for component presence/absence, orientation and alignment, and postsolder inspection for solder integrity. The desire for automated inspection in these areas is illustrated by the following points:

- a requirement for improved yield; e.g. a large computer manufacturer is installing a vision machine to inspect the underside of its small system mother boards to check the presence of chip capacitors prior to flow solder.
- a requirement to provide statistical process control information, typically to ensure the solder paste printing process is optimised. Another large computer manufacturer is as yet unable to find suitable equipment.
- a requirement for guaranteed long-term reliability where solder joint integrity inspection is a must, e.g. space, defence, nuclear, aerospace and computer industries.

Vision machine requirements for inspection during assembly of PCBs

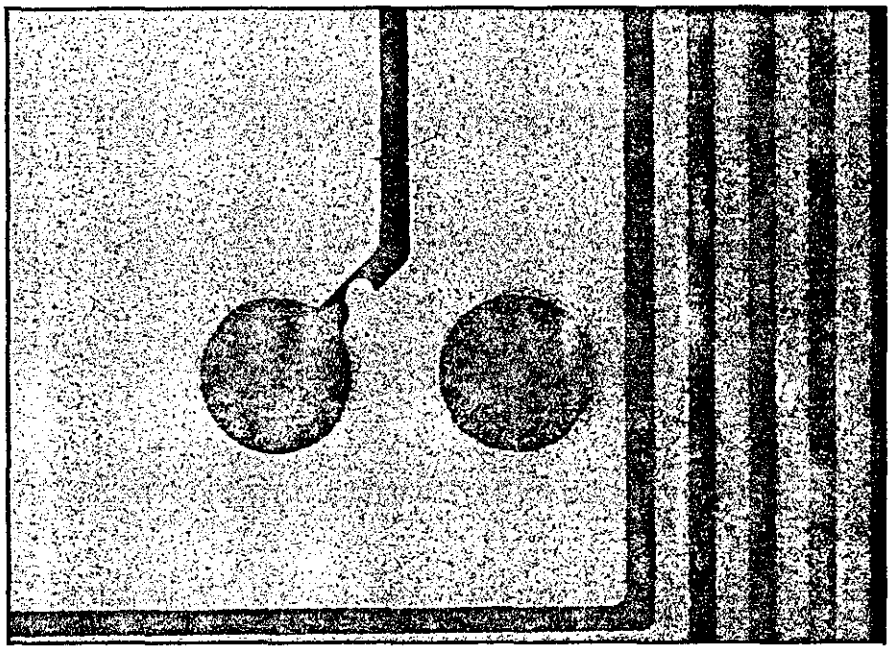
The following typical vision machine requirements can be applied to most inspection applications. Through considering these requirements, generalised observations can be determined.

☐ Illumination — vision machines cannot easily adapt to variable lighting conditions, but as well as providing a consistent lighting situation, the method of illuminating can be such that the pertinent features of the scene being viewed are enhanced. This can minimise subsequent image processing and increase the system speed, e.g. backlighting of components for component alignment/inspection immediately provides the required component outline. However, no such simple solution exists for general inspection to improve PCB assembly processes.
☐ Resolution — when using a vision machine to produce alignment information in order to improve the placement

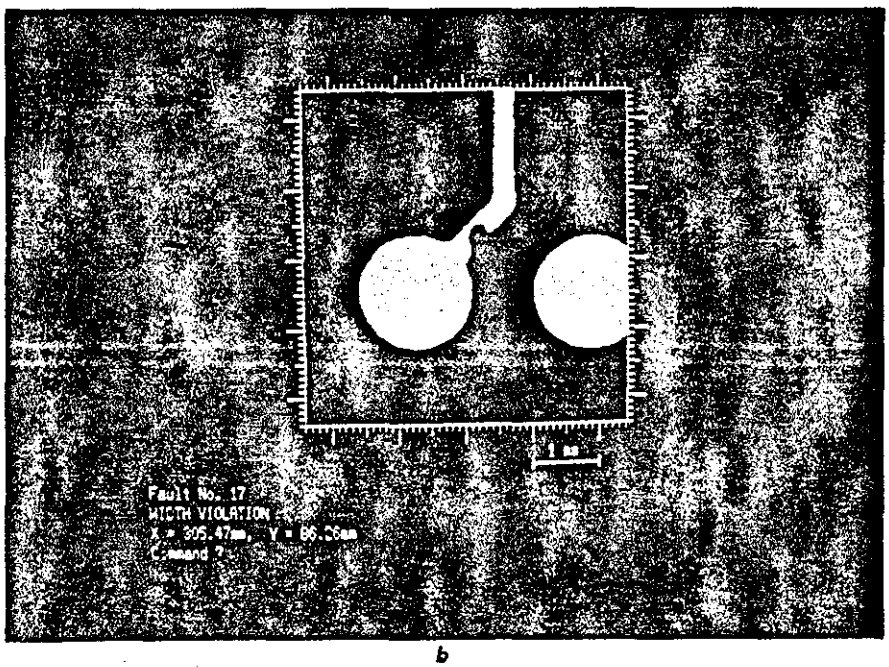
accuracy (e.g. of an FPT component with a lead width of 0.2mm on a pitch of 0.5mm), ideally the vision machine is required to have a spacial resolution of approximately an order better than the lead pitch [3] (circa 0.05mm in this case). A vision machine inspecting satisfactory alignment prior to soldering must also provide similar resolution.

☐ 3-D range information — in application areas such as artwork inspection the information required is of a two-dimensional nature so conventional 2-D image analysis techniques can be employed. However, other applications, typically solder paste inspection prior to component placement, require depth information [4], necessitating 3-D vision machines.

☐ Speed — throughput is clearly a major consideration when choosing PCB assembly methods. Having invested millions of pounds in high-speed automated assembly lines, it is essential not to limit their potential by incorporating slow inspection stages or to slow down an onsertion machine while its vision machine struggles to locate a fiducial mark. It is equally important not to engage in the high-speed production of scrap. Historically, vision machines have been slow, principally due to the large amounts of data required to be processed. The necessity of applying vision to time critical operations has led to techniques being developed to get over the problem. In addition, enabling technology has provided ever faster processing platforms and specialist vision hardware; typically array processors for fast implementation of neighbourhood processing algorithms, and dedicated hardware for even faster implementation of commonly used image-processing algorithms (e.g. edge trace, edge match) [5]. However, implementation in hardware can mean a consequent loss of flexibility.



a



b

Fig. 5 Images of PCB tracks and pads of *a* the original PCB and *b* a binary image captured using Trackscan 3000 equipment identifying the track width violation.
Courtesy of Lloyd Doyle Ltd.

Speed is very important if 100% automated inspection is required where it is necessary to inspect as fast as the controlled machine (or indeed operator) can assemble. However, complex operations such as 3-D inspection of solder joints are time-consuming. It is also important to stress that this type of inspection information is of great value to feed back to control previous process variables or as part of a statistical process control scheme [6], but does not necessarily take place on 100% of the product (i.e. with current enabling technology, achievable speeds are such that machine vision inspection during PCB assembly can only be used in controlling loosely coupled processes).

Illumination, resolution, speed and 3-D are typical requirements which suppliers have addressed and, in some cases, successfully implemented. The more general requirements, of a highly automated industry trying to increase quality and efficiency, which must be applied to vision machines are flexibility and consistency or classification reliability. The experiences of three users of machine vision for PCB assembly inspection illustrate this.

Typical experiences of industry

Experiences in the UK

A large computer manufacturer with production plant in the UK is looking to machine vision to provide automated inspection in a number or areas, including final inspection of SMT boards prior to electrical test. They fear similar problems to those experienced on their through-hole technology, final PCB automated, visual inspection equipment.

This equipment comprises a four-camera vision machine viewing the underside of finished PCBs, illuminated using banks of LEDs. Basically the system is looking for a high intensity of reflected light from the clinched lead to confirm all leads are through holes. The teaching procedure is to define windows at each lead site and then, by using a good board, set up the pass/fail criteria. The view from each camera must be inspected and a priority system set up according to the usefulness of each view in classifying pass or fail. Judging from past experience, having run the system for several years, a thousand lead PCB will take at least two

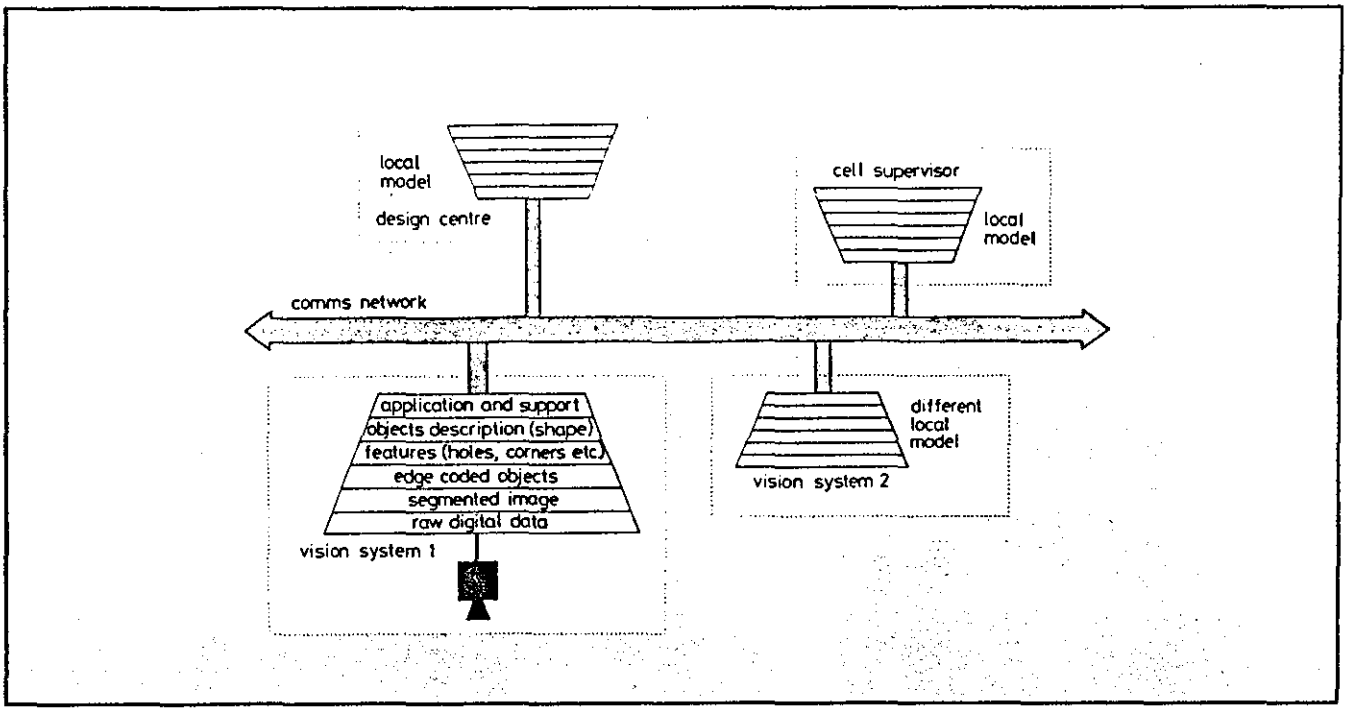


Fig. 6 · A possible hierarchical model for vision system integration.

days of hard laborious work to set up.

Having set up the system, the consistency of the pass/fail classification is often poor. Slight changes in production profiles, such as lead clinch angle, lead crop height or height of the flow solder wave, can alter the appearance of the expected image in each window, and the vision system cannot adapt. The system works but requires the regular attention of a technician to adjust the vision machine, providing the feedback between process variations and inspection criteria. The technician transfers information between programmable machines.

Experiences in the USA

Some companies in the United States have experience of the use of vision machines for inspection during SMT PCB assembly. The fears of the UK manufacturer were borne out by a large consumer electronics manufacturer in the US who, in recent years, has applied a vision machine for presolder inspection, primarily checking for presence/absence and correct alignment.

On a line producing TV tuner circuits, incorporating over 600 components with an assembly cycle time of 20 seconds, the company spent over two years tuning the vision system in an attempt to achieve 100% inspection with an acceptable fault classification reliability.

The inspection system used a single moving camera with a strobed fluorescent ring light arrangement, producing a 256 grey level 2-D image, supported by powerful processing equipment to achieve image analysis and pass/fail classification at high speed. The company were unable to achieve satisfactory classification reliability, primarily due to the complexity of the problems in setting up the system to allow for changing process variables; typically, optical properties of the same components varied, optical properties of solder paste, pad and PCB composition meant setting up times of two to three days followed by regular adjustment.

Experience gained during the operation of the vision equipment allowed the users to modify their assembly process, resulting in improved quality, such that the limited benefits to be gained from presolder inspection within a mature production process meant that it was no longer viable, in its present form, for their production scenario.

The company now sees the role of vision inspection as a low-speed offline operation providing process control information to ensure the assembly process remains tuned for high quality production.

The experiences of another manufacturer, using a similar vision machine, demonstrates what can be achieved in the field of automated inspection during PCB assembly. Genuine benefit can be derived from the inspection process and the development resource is available to tailor the inspection system to the specific requirements of a particular manufacturing process and to incorporate the inspection system within a flexible manufacturing environment.

The company in question is a large US-based computer company with an entirely different production scenario to the previous company, a production rate of 1 000 boards per day and a mix of 150 board types with 30% SMT, 20% mixed and 50% through-hole technology. Two years ago they employed eight inspectors per shift for presolder inspection of placed components from their three SMT lines, checking for 75% leg on pad component alignment. Problems included speed, operator fatigue/inconsistency and a tendency to adjust for leg on paste rather than pad.

In 1987 vision inspection equipment was incorporated in the SMT assembly process for presolder alignment inspection. Then followed a two-year period of development, involving modification of the strobed ring light illumination and camera optics, and extensive trials and tuning in order to generate a satisfactory set of inspection classification algorithms so that the equipment should operate reliably. This work was carried out by an engineer with previous vision system experience and a close working relationship with the software development staff at the vision equipment manufacturers.

The vision system inspects the position of component legs against expected position data (much as the human inspectors had done). The algorithms used to process the vision information and acceptance criteria for pass/fail decision making are based on the requirements of individual device types. At present six types have evolved:

- SOIC style components with gull wing leads on 1.27mm pitch to two sides of the component;
- PLCC style components with j leads on 1.27mm pitch to four sides of the component;
- two-terminal chip style components (e.g. chip capacitors);
- SOIC style components in white, with gull wing leads as above;
- j-leaded components with 1.27mm pitch leads to two sides of the component;
- SOIC style components as above with 0.635mm lead pitch.

A suite of auto program generation software has been written by the computer manufacturer's software staff, to enable CAD data to be used along with the library of six algorithms in order to generate the vision machine software required to inspect a new or modified PCB. The programmers responsible were advantaged by their experience of UNIX and the vision machine's use of REGULUS (real-time UNIX). A level of flexibility has been introduced specific to this equipment, providing offline vision system software generation without the use of 'golden boards' or matching techniques.

Two years ago 30% of components required adjustment prior to reflow solder. The inspection equipment is now used during production line set up to adjust offsets on the three onsertion machines, and a reduction to 10% component adjustment has been achieved. Reliability of fault classification has been improved to 98% genuine faults.

100% inspection of SMT PCBs takes place. Boards are queued in batches from each onsertion machine, requiring the inspection equipment to be reprogrammed up to ten times per day. Following inspection, PCBs are routed to two rework stations which are data-linked to the inspection system, fault data is downloaded and misplaced components are identified, using light pointing, for realignment by the two inspectors now required to man the line. The success of the system can be attributed to both its ability to achieve high classification reliability and the degree of flexibility provided by the integration of the system within the software tools and programmable machines comprising the design and manufacturing process. The ability to offline program/teach the vision machine, minimising set up times during product changeover, is of great importance within this type of production scenario.

Machine vision integration

In the previous example the computer manufacturer was able to make use of dedicated inhouse vision and software expertise. Ideally vision machines, like any other manufacturing tool, should be easy to install, modify and support. This implies a need for standard and consistent approaches, but the problems are considerable:

- a significant variety of potential application areas exist;
- there is a wide choice of processing algorithms, illumination techniques, sensors etc., and their implementation is very vendor specific (i.e. choice of hardware — processors and specialist electronics — and software — data structures, languages and operating systems);
- standardisation implies an overhead, which could increase processing times and costs.

Vision machines must also cope with product variation. The trend towards discrete batch manufacture implies a need to

- ☐ reduce set up/changeover times;
- ☐ monitor the process to improve the planning and scheduling of manufacture.

Both ideally require data-driven automation, implying a high degree of integration, i.e. reuse of design information (e.g. process or product models), in minimising teaching cycles.

These implications suggest a requirement for flexibility within machine vision. It is a paradox that vision machines being software-based are inherently very flexible, yet their current use in manufacturing equates closer to hard automation, each application being relatively inflexible and dedicated to a limited range of uses.

When considering the interfacing of a generic vision machine, as briefly described above, there is the potential for providing a range of levels of information from the raw grey level pixel array to analysed image data (Fig. 6). The machine could also accept information from other manufacturing entities at the same set of levels. This set of levels could be used as the basis of a structured hierarchy for vision information within the vision machine and other manufacturing entities to enable structured integration of vision within manufacturing systems in a flexible and consistent manner [7].

Conclusions

In today's increasing climate of product variety and subcontract manufacture the necessity for flexibility and reliability, leading to short changeover times and consequent reduced plant downtime, is paramount.

Turnkey vision systems are established in specific areas where economics and engineering constraints have demanded that vision be made to work both reliably and fairly flexibly. Although not trivial, natural features of these applications have contributed greatly to their success. Success in other applications, such as inspection during assembly, is proving more difficult to achieve.

The flexibility provided in the successful applications has typically been achieved by providing links from groups of shop-floor machines to a supervisory system, providing facilities such as software libraries for simplifying the changeover or teaching operations. In terms of integration all this provides is a bigger closed system, rather than a collection of programmable manufacturing entities, communicating using a structured hierarchical approach, which could then be integrated with other manufacturing entities throughout the manufacturing plant (i.e. the design office).

The controlled integration of vision machines into CIM systems could provide an environment in which some of the problems of flexibility and reliability may be solved. Benefit may be gained by teaching vision machines using offline programming techniques, making use of design and modelling information created on CAD systems. A theoretical model of a particular inspection feature could be modified, depending on process variables. The necessity to modify inspection criteria dependent on process changes will only be possible when access to both design and process information is available to the supervising systems controlling the vision machines. The reuse of this information will only become possible when it is controlled under standardised architectures.

References

[1] WESTON, R. H., *et al.*: 'Steps towards information integration in manufacturing', *Int. J. Comput. Integr. Manuf.*, 1988, **3**, pp. 140–153

[2] STROEBEL, T.: 'Fine pitch SMT assemblies: emerging inspection issues', *Printed Circuit Assem.*, 1989, **3**, pp.21–27

[3] BLACK, R. J.: 'Versatility and vision: the key to accurate placement of future surface mount packages,' *Electron. Produkt. Pruftech.*, 1988, **8**

[4] VOSS, W. D., *et al.*: 'Measure solder paste in the third dimension', *Electron. Packag. Prod.*, 1989, **1**, pp. 42–44

[5] OUGHTON, D. M. A.: 'Automated placement of fine lead pitch surface mount electronic devices using vision guidance', *Adv. Manuf. Eng.*, 1988, **1**, pp. 6–10

[6] NAMETH, R.: 'How to use automated X-ray inspection for process monitoring and control', *Solder. Surf. Mount Technol.*, 1989, **1**, pp. 47–51

[7] AZAR, I., *et al.*: 'On vision architecture for computer integrated manufacturing', *Microprocess. Microsyst.*, 1988, **1**, pp. 24–32

John Edwards is with the Department of Manufacturing Engineering, Loughborough University of Technology, Loughborough, Leicester LE11 3TU, UK.