# Reference architecture for configuration, planning and control of 21st century manufacturing systems.

LASSILA, Anna-Maija.

# REFERENCE

# Reference architecture for configuration, planning and control of 21st century manufacturing systems

Anna-Maija Lassila

A thesis submitted in partial fulfilment of the requirements of
Sheffield Hallam University
for the degree of Doctor of Philosophy

June 2007

Collaborating organisation: EPSRC

# Preface

This thesis is presented as part of the requirements for the award of the degree of Doctor of Philosophy from Sheffield Hallam University in the UK. It reports on the research work carried out by the author under the supervision of Prof. Sameh Saad in the Faculty of Arts, Computing, Engineering and Sciences at Sheffield Hallam University between October 2001 and June 2007. The research was financially supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under the research grant GR/R00432/01.

The oral examination of the thesis was conducted by Prof. Nabil Gindy from the University of Nottingham and Prof. Terrence Perera from Sheffield Hallam University on 26 July 2007.

Candidate's declaration of originality:
This thesis is my own work and contains no material which has been accepted for the award of any other degree or diploma in any university and, to the best of my knowledge and belief, it contains no material previously published by any other person except where due acknowledgement has been made.

Anna-Maija Lassila
June 2007

## Acknowledgements

I have received invaluable support, encouragement and guidance from many people during the lengthy period of preparing this thesis. The enthusiasm of colleagues, family and friends towards my research was a vital source of motivation on those moments when frustration and discontent threatened the progress of this project. I would like to express my gratitude for everybody who has supported me during this momentous phase in my life. I would have never been able to do this without you.

The support and guidance of my project supervisor Prof. Sameh Saad has been a key to the success in completing this thesis. His critical and constructive advice throughout the project has been significant for the progress and quality of this research. I would also like to thank him for the many opportunities he has offered me to improve my academic record in terms of publications, attending international conferences and working in national and international academic and industrial projects. Furthermore, I express my gratitude to the UK Engineering and Physical Sciences Research Council (EPSRC) for funding this research programme.

Finally, I would like to thank my entire family for their encouragement and understanding. I would especially like to thank my husband Benjamin, who will probably be more proud of my new academic title than I will, for his love, friendship and support. You have brought sunshine to my life. I would also like to express my appreciation to my parents for their support throughout my life and dedicate this thesis for them.

*Tämä väitöskirja on omistettu*
*Äidille ja Isälle*
*kiitokseksi saamastani tuesta ja kannustuksesta.*

## Abstract

Today's dynamic marketplace requires flexible manufacturing systems capable of cost-effective high variety - low volume production in frequently changing product demand and mix. Several new paradigms, e.g. holonic, fractal, biological and responsive manufacturing, have recently been proposed and studied in the academic literature. These 'next generation of manufacturing systems' have been especially designed to meet the requirements of an unstable and unpredictable marketplace. However, very little in-depth research of the configuration, planning and control methodologies of these new concepts has been conducted. This research aims to improve the comprehension and implementation of these 21st century manufacturing systems by developing an integrated reference architecture from the combination of their distinctive features that would enable manufacturing enterprises to handle successfully the configuration/reconfiguration, planning and control activities under the conditions of uncertainty and continuous change.

In the course of the research, a detailed investigation into the fractal, biological and responsive manufacturing systems is conducted in order to identify the strengths and weaknesses of each concept. The common and distinctive features of the paradigms are then used to merge them to create an integrated reference architecture. The fractal configuration, biological scheduling and 'resource element' representation of resource capabilities and product processing requirements are selected as the major elements of the new system. A detailed study of fractal layout design resulted in seven distinctive methods for structuring and managing fractal cellular systems. A design methodology that supports three types of dynamic scheduling is developed for biological manufacturing systems. Resource elements are used with fractal layouts and biological scheduling to enhance performance and to enable an integration of the concepts. The proposed reference architecture is modelled and evaluated using object-oriented programming, computer simulation and heuristic algorithms. The research results indicate that the performance of systems that employ biological scheduling and fractal layouts can be improved by using the concept of resource elements to utilise any hidden capabilities of resources and to achieve an optimal distribution of resources on the shop floor.

## Publications from the thesis

Saad, S. M. and Lassila, A. M., 2006, An integrated approach for shop floor configuration in fractal manufacturing systems. *International Journal of Services and Operations Management,* 2, 2, 109-123.

Saad, S. M. and Lassila A. M., 2005, Configuration of manufacturing resources using the fractal cell concept. *Proceedings of the 3rd International Conference on Responsive Manufacturing (ICRM2005).* 12-14 September 2005, Guangzhou, China.

Saad, S. M. and Lassila, A. M., 2005, Decentralised scheduling in biological manufacturing systems. *Proceedings of the 15th International Conference on Flexible Automation & Intelligent Manufacturing (FAIM2005).* 18-20 July 2005, Bilbao, Spain.

Saad, S. M. and Lassila, A. M., 2004, Methodology for distributed real-time scheduling and control in biological manufacturing systems. *Proceedings of the 8th International Conference on Manufacturing & Management (PCMM2004).* 8-10 December 2004, Gold Coast, Australia.

Saad, S. M. and Lassila, A. M., 2004, Flexibility study of fractal layouts - Part 2. *Proceedings of the 2nd International Conference on Manufacturing Research (ICMR2004).* 7-9 September 2004, Sheffield, UK, pp. 209-215.

Saad, S. M. and Lassila, A. M., 2004, Layout design in fractal organizations. *International Journal of Production Research,* 42, 17, 3529-3550.

Saad, S. M. and Lassila, A. M., 2004, Flexibility study of fractal layouts - Part 1. *Proceedings of the 14th International Conference on Flexible Automation & Intelligent Manufacturing (FAIM2004).* 12-14 July 2004, Toronto, Canada.

Saad, S. M. and Lassila, A. M., 2003, Layout design in fractal organisations. *Proceedings of the 17th International Conference on Production Research (ICPR-17).* 3-7 August 2003, Blacksburg, Virginia, USA.

# Table of contents

# List of tables

# List of figures

# Nomenclature

$c_f$ fractal cell $f$,

$c_s$ specialised fractal cell,

$D$ total demand,

$D_j$ demand of product type $j$,

$D_f$ demand at fractal cell $f$,

$D_i$ demand at any fractal cell $i$,

$D_{fj}$ demand of product type $j$ at fractal cell $f$,

$d$ single product,

$F$ number of fractal cells,

$F_R$ number of non-specialised fractal cells,

$G$ sample difference,

$J$ number of different product types,

$l_{fj}$ number of leftover products of type $j$ allocated to fractal cell $f$,

$l_{fm}$ number of leftover machines of type $m$ allocated to fractal cell $f$,

$M$ number of different machine types,

$m$ machine type $m$,

$m_s$ machine in the specialised fractal cell $c_s$,

$N$ total number of machines,

$N_m$ number of machines of type $m$,

$N_f$ number of machines in fractal cell $f$,

$N_{fe}$ number of empty machine fields in fractal cell $f$,

$N_i$ number of machines in any fractal cell $i$,

$N_s$ number of machines in specialised fractal cell $c_s$,

$N_{fm}$ number of machines of type $m$ in fractal cell $f$,

$r$ total number of routing events between machines to process $D$

$R_f$ size of fractal cell $f$ in number of machine fields,

$S$ total travelling distance ($S = S_{\text{int}} + S_{\text{coop}}$),

$S_{\text{coop}}$ total cooperative travelling distance,

$S_{\text{coop},d}$ cooperative travelling distance for product $d$,

$S_{\text{int}}$ total internal travelling distance,

$S_{\text{int},f}$ internal travelling distance for all products in fractal cell $f$,

$S_{\text{int},fd}$ internal travelling distance for product $d$ in fractal cell $f$,

$S_{proc}$ total internal and cooperative travelling distance ($S_{proc}= S_{int}+ S_{coop}$),

$S_w$ weighted total travelling distance to reflect statistical differences in $r$

$T_{jm}$ processing time of a product type $j$ at machine $m$,

$U_m$ time availability of machine $m$,

$x$ horizontal dimension of a cell (in machines ) or the shop floor (in cells),

$x_f$ horizontal dimension of cell $c_f$,

$y$ vertical dimension of a cell (in machines ) or the shop floor (in cells),

$y_f$ vertical dimension of cell $c_f$,

$Z_A$ size of tabu search neighbourhood.

# Chapter One - Introduction

This chapter introduces the research problem to be addressed. It includes a review of the research background, rationale and purpose. It also summarises the research objectives and the expected outcomes. Later in the chapter, the research questions that form the foundation for the investigation are formulated. The chapter closes with an outline of the thesis.

## 1.1 Background and rationale

The social, political and economic changes of the last two decades have resulted in an increasingly volatile and unpredictable business environment characterised by a shortening of product life cycles, diversified customer taste, increasing demand for customised products, global competition and frequent changes in fashion and technology (Clark and Fujimoto 1991, Levary 1992, Sharifi and Zhang 1999). Manufacturing in this dynamic environment can be a challenging task as it requires constant adaptation and rapid response (Balasubramanian *et al.* 2001). The traditional manufacturing systems designed for stable long-term production of only a few product variants are often unable to cope efficiently with the frequent changes in process requirements and production orders (Bongaerts *et al.* 1997a, Koren *et al.* 1998). On the other hand, the capabilities of lean manufacturing to handle disturbances have lately been questioned in the academic literature, e.g. Suda (1989), Cusumano (1994), Katayama and Bennett (1996) and Gould (1997). In addition, Small and Downey (1996) and Yang *et al.* (2004) noted that turbulent times and uncertainty in the business environment are the main causes of failure in manufacturing industry. Hence, new more flexible manufacturing systems capable of cost-effective high variety and low volume production with frequently changing product demand and mix are required.

As a result, several new paradigms for the design, planning and control of manufacturing systems have recently been proposed and studied in the academic literature. They include some acclaimed concepts such as agile (e.g. Gunasekaran 1998, Sharifi and Zhang 2001), holonic (e.g. Van Brussel *et al.* 1998, McFarlane and Bussmann 2000), fractal (e.g. Warnecke 1993, Venkatadri *et al.* 1997), biological (e.g. Ueda *et al.* 1997b, Ueda *et al.* 2001a), virtual (e.g. Irani *et al.* 1993) and responsive (e.g. Gindy *et al.* 1996, Saad and Gindy 1998) manufacturing. These theories possess many similar features such as autonomy of their basic units, distributed structures and cooperation (Tharumarajah *et al.* 1996). These 'next generation manufacturing systems' (also known as '21st century manufacturing systems') have been specifically designed to meet the requirements of an unstable and unpredictable marketplace (Wyns 1999). They have been claimed to offer more flexible, responsive and adaptable structures and processes than the traditional manufacturing approaches (Tharumarajah *et al.* 1996, Sousa *et al.* 1999). Although the demand for and the potential of these new theories

have been widely recognised in the academic literature, very little in-depth research of their design methodologies and capabilities have been conducted (Sun and Venuvinod 2001). In addition, their deployment in the industry is almost nonexistent due to the abstract nature of the concepts and the lack of clear implementation procedures. Thus, it is time to investigate the features and processes of these emerging manufacturing systems.

So far, the reported research on 21st century manufacturing systems has predominantly addressed the paradigms in isolation focusing each time on only one specific area of operations management. Examples of this are shop-floor configuration in fractal (e.g. Venkatadri *et al.* 1997, Montreuil *et al.* 1999) and biological (e.g. Vaario and Ueda 1997) organisations and scheduling in holonic (e.g. Bongaerts *et al.* 1997b, Sousa and Ramos 1998), biological (e.g. Vaario and Ueda 1998a) and agile (e.g. He *et al.* 2001) manufacturing systems. This approach has lead to a fragmented research with partly incompatible views on the basic characteristics of the concepts and their application. The reference architectures proposed by Wyns (1999) for holonic and Ryu and Jung (2003) for fractal manufacturing systems provide more comprehensive visions of these theories but lack clear implementation procedures. The ongoing intelligent manufacturing system (IMS) research programme of the next generation of manufacturing systems (NGMS) differs from any previous and subsequent research in its strategy of studying multiple concepts simultaneously, namely agile, fractal, biological, and autonomous & distributed manufacturing. It attempts to merge them into a unified framework that covers all product-related functions of a global enterprise from production to logistics and post-sales services (NGMS-IMS consortium 2000). Although the idea of integrating multiple concepts into a single system in order to utilise the most innovative features of various theories simultaneously is viable and promising, the extensive scope of the NGMS-IMS project complicates the resulting system and limits the level of detail it can contain. In addition, an integration of the paradigms takes place on operative borders only, which simplifies the application of the concepts but prevents full exploitation of their capabilities. To truly benefit from the distinctive features of several paradigms, the integration of concepts should occur at the functional level, for example in factory floor operations. Up until now only a few research papers with limited scope have addressed this issue, e.g. Saad (2003) merged

virtual and responsive concepts to develop a procedure for the reconfiguration of cellular manufacturing systems.

The idea of constructing a new system by combining several highly advanced concepts is appealing as it potentially offers a simple and fast approach to system development and enables the simultaneous exploitation of distinctive beneficial features of several concepts. However, for the integration to succeed it is important to first carefully define the scope of the new system and the performance objectives. The mix of concepts to be merged should then be selected so that i) they have similarities that enable the integration, ii) they posses one or more distinctive features that support the objective and are applicable within the scope of the system, and iii) when combined they cover all areas within the scope of the system. As this requires detailed knowledge about the strengths and weaknesses of each concept and their common and distinct features, a comprehensive analysis and comparison of the emerging manufacturing paradigms is vital. The assessment of fractal, holonic and biological concepts has been conducted by Tharumarajah *et al.* (1996, 1998), Kadar *et al.* (1998), Sousa *et al.* (1999) and Ryu and Jung (2003), who noted that the underlying principles of the paradigms are very similar. Based on these reports and the studies by Venkatadri *et al.* (1997), Montreuil *et al.* (1999) and Askin *et al.* (1999) in the fractal layouts, by Vaario (1996), Ueda *et al.* (1997b, 1998), Fujii *et al.* (1997), Vaario *et al.* (1997) and Vaario and Ueda (1998a) in the dynamic scheduling in biological organisations, and by Gindy *et al.* (1996, 1999), Gindy and Saad (1998) and Saad and Gindy (1998) in the responsive manufacturing concept of presenting machine capabilities and product processing requirements as resource elements, fractal, biological and responsive paradigms are chosen in this research for integration within the scope of shop floor operations, i.e. layout design, scheduling and control. The objective of the integrated system is to enable rapid and cost-effective response to changes in the manufacturing environment.

## 1.2 Purpose

The purpose of this research is to develop an integrated reference architecture for configuration, planning and control of the 21st century manufacturing systems in order to facilitate efficient production in today's volatile and unpredictable business environment. In this context, a reference architecture can be defined as a generalised

model or template that outlines the principles and rules for system development in a specific domain (Wyns *et al.* 1996).

### 1.2.1 Research objectives

The principle aim of this research is to enable manufacturing enterprises to adapt and find a rapid and balanced response to customer requirements under conditions of uncertainty and continuous change. To meet this challenge a number of specific objectives are identified and summarised as follows:

a) to identify the features of emerging manufacturing concepts,
b) to provide a comprehensive comparison of the design, operational and organisational characteristics of these paradigms (namely fractal, holonic and biological manufacturing systems) in order to identify the common and distinctive features of the concepts,
c) to develop a new integrated reference architecture from the combination of responsive, biological and fractal concepts to successfully handle the configuration/reconfiguration, planning and control activities, and
d) to evaluate the proposed architecture using modelling and computer simulation

### 1.2.2 Expected outcomes

This research is expected to yield a significant contribution to the design, implementation and integration of fractal, biological and responsive manufacturing systems. The expected outcomes can be summarised as follows:

a) a report on the characteristics of the 21st century's manufacturing systems,
b) a comprehensive comparison of the design, organisational and operational features of these concepts,
c) a framework for the integration of fractal, biological and responsive manufacturing systems,
d) a methodology for shop floor configuration in fractal manufacturing systems,
e) a procedure for real-time scheduling in biological manufacturing systems,
f) a process for representing resource capabilities and product processing requirements as resource elements,
g) a mechanism for incorporating resource elements into fractal layout design and biological scheduling, and

h) an evaluation of the proposed integrated reference architecture using experimental data obtained through computer simulation

## 1.3 Research questions

The main issues addressed in this study can be summarised in three questions, which form the basis of the research:

Question 1:

*What are the distinctive and common features of the emerging manufacturing concepts?*

Question 2:

*What benefits can be achieved through the application of these new paradigms?*

Question 3:

*To what extent is it possible and beneficial to integrate some of these concepts in order to exploit their distinctive characteristics regarding the production planning and control activities, and how could this be done?*

In order to answer these research questions the study concentrates on identifying and exploiting the capabilities of the new manufacturing theories. The research focus will be defined in more detail at the end of the next chapter after relevant literature has been reviewed.

## 1.4 Outline of the thesis

The conducted research can be divided into three main phases: i) review and comparison of the emerging manufacturing systems, ii) development of an integrated reference architecture, and iii) evaluation of the proposed reference architecture. This work is documented and presented in this thesis in eight chapters as follows:

Chapter 1 introduces the research problem that is to be addressed. It includes a review of the research background, rationale and purpose. It also summarises the research objectives and the expected outcomes. Later in the chapter, the research questions that form the foundation of the study are formulated. Finally, an outline of the different chapters in the thesis is provided.

Chapter 2 provides an overview of the theory and recent academic research regarding the traditional manufacturing systems and the emerging manufacturing paradigms. The chapter begins with a short review of the history of manufacturing management and an assessment of the reasons behind today's volatile business environment. Next, reference architectures are introduced and the traditional manufacturing layouts and production planning and control techniques are discussed. The later half of the chapter is dedicated to the review and analysis of the next generation of manufacturing systems. Finally, a comprehensive comparison of the design, operational and organisational features of fractal, biological and holonic manufacturing concepts is conducted. The chapter concludes with the formulation of the research focus.

Chapter 3 describes the methodology employed to answer the research questions. First, the general approach for solving the research problem is determined and justified. Next, the major steps in the research process are summarised. The most important tools and techniques deployed during the study, including mathematical and conceptual modelling, heuristic algorithms and computer simulation are reviewed and evaluated. Finally, a hypothetical case study for the implementation, testing and analysis of the developed reference architecture is introduced.

Chapter 4 reports on the development and application of the responsive manufacturing paradigm in the context of dynamic self-organisation-based scheduling systems. First, the relationship between manufacturing flexibility and responsiveness is examined. Then the potential for flexibility in the resource-element-based representation of machine capabilities and product processing requirements are investigated. Subsequently, a procedure for the application of resource elements in self-organisation-based scheduling is proposed. The methodology is then modelled and simulated using Arena simulation software. Finally, a number of experiments are conducted based on the hypothetical case study.

Chapter 5 proposes a methodology for shop floor configuration in fractal manufacturing systems. It includes an overview of the layout design process and a discussion on system re-configurability. It also summarises the existing fractal layout design approaches. The chapter goes on to propose seven distinct fractal cell configuration

methods for different system design objectives and constraints, and to develop an integrated design methodology. In addition, mathematical models applicable to the different design stages are presented. The procedure is modelled and simulated with heuristic algorithms deployed using c++ programming language. Finally, the procedure is applied to the hypothetical case study. The quality of the resulting layouts is assessed and compared against a random arrangement of machines. The chapter ends with a study on the flexibility of these layouts.

Chapter 6 suggests a methodology for dynamic self-organisation-based distributed scheduling and control in biological manufacturing systems. First, the static and dynamic scheduling approaches and centralised and distributed control techniques are reviewed. Then the exiting biological scheduling methods are summarised. This is followed by a proposal for a biologically-inspired decentralised real-time scheduling methodology. Three types of dynamic scheduling are then identified and operational procedures are developed. Finally, the proposed methodology is modelled and simulated using Arena simulation software and applied to the hypothetical case study.

Chapter 7 develops an integrated reference architecture for the configuration, planning and control of the 21st century manufacturing systems. A framework for the integration of fractal, biological and responsive concepts is developed based on the conducted research into fractal layouts, biological scheduling and control, and resource elements. The interconnections between these concepts under a unified framework are investigated and defined. After that, the proposed framework is generalised to formulate a universally applicable reference architecture for shop floor operations. Then the proposed architecture is modelled and simulated using Arena simulation software. Finally, the architecture is evaluated by applying it to the hypothetical case study.

Chapter 8 provides the concluding discussions of the research. It includes a review of the relevant literature and the conducted research. It also comprises a discussion of research findings and their broader implications. The limitations of the research methods and findings are then discussed. Next, the contributions of the research to knowledge in the field of the 21st century manufacturing systems are summarised. The chapter ends with some suggestions for the future work.

# Chapter Two - Literature review

In this chapter, an overview of the theory and recent academic research regarding the traditional layout design and production planning and control methodologies and the next generation of manufacturing systems is presented. The chapter starts with a short review of the history of manufacturing and operations management. The ongoing debate over the lean manufacturing methods and the rapidly changing global business environment are identified as the main reasons behind the emergence of new manufacturing concepts. Next, the reference architectures are introduced and the limitations of the traditional manufacturing layouts and production planning and control techniques are discussed. The later half of the chapter is dedicated to the review and analysis of the 21st century manufacturing concepts, which are often referred to in the academic literature as the next generation of manufacturing systems. Finally, a comprehensive comparison of fractal, biological and holonic manufacturing concepts is carried out in order to identify the distinctive and common features among these paradigms that might enable their integration. The chapter ends with the formulation of the research focus.

## 2.1 Evolution of manufacturing systems

The Industrial Revolution is generally regarded as the beginning of modern-style manufacturing. Prior to it, production was predominantly conducted by hand on a small scale for limited markets. The transformation from traditional handicraft production to modern industrial systems started in the English textile industry in the mid eighteenth century when a series of technological innovations was introduced in favourable socio-economic conditions (Mathias 1983). The innovations that enabled the substitution of labour for machinery generated for the first time economies of scale that made large-scale production in centralised locations attractive i.e. creating the factory system (Hopp and Spearman 2000). Amongst the most important innovations of the early industrial development were the steam engine, the notion of division of labour, and the concept of interchangeable parts. Yet, the full potential of the latter two concepts was not utilised until one hundred years later when the manufacturers in the United States embraced the concept of mass production.

Over the course of the nineteenth century industrialisation spread from England to continental Europe and North America. In the United States the rapid expansion of the transportation network, e.g. railway and steamship, created a mass distribution system that in the end of the century stimulated a revolution in mass production technology (Hopp and Spearman 2000). As the volume of production and the size of factories grew, the complexity of problems in the organisation, coordination and control of the manufacturing processes increased (Duguay *et al.* 1997). This resulted in the rise of scientific management (Taylor 1911), a managerial approach that focused on the systematic organisation, optimisation and standardisation of work. In 1913 the introduction of the moving assembly line extended the benefits of high-speed mass production from process industries to the manufacturing of complex mechanical products such as cars (Hopp and Spearman 2000). Duguay *et al.* (1997) summarised the main characteristics of the traditional mass producers as follows: i) cost reduction is obtained by increasing the volume of production, ii) the production system is improved through major innovations, iii) direct labour executes tasks under the supervision of managers, and iv) suppliers are made to compete against each other.

The American system of mass production was the leading market force until the 1970's when finally the European and Japanese manufacturers had fully recovered from the Second World War and were able to compete with American companies. The new competitive pressures exposed limitations of traditional mass production, i.e. inflexibility, large inventories, long output times, process instability, and extensive quality assurance (Bullinger *et al.* 1995), and led to a decline of American industry. To reverse this trend a number of technological, organisational and managerial innovations were proposed. They included concepts such as lean manufacturing, total quality management, concurrent engineering, business process reengineering, management by objectives, world class manufacturing, manufacturing resource planning, statistical quality control, computer aided design and manufacturing, computer integrated manufacturing, intelligent manufacturing systems, flexible manufacturing systems, just-in-time, etc. Although the value of some of these innovations have been questioned in the academic literature (e.g. Sharifi and Zhang 1999, Hopp and Spearman 2000), in general, their positive influence on the development of the mass production paradigm have been recognised (e.g. Doll and Vonderembse 1992, Bartezzaghi 1999).

In the 1950's the Toyota Motor Company started to develop a distinctive style of manufacturing. Instead of obtaining low costs through high volume, which was not an option on the limited Japanese domestic market, the company focused on enabling high model variety production through flexible processes (Hopp and Spearman 2000). The new technique involved creating a smooth production flow, use of self-monitoring machines, a flexible workforce, and the elimination of inventories, rework and set-up times by implementing just-in-time and continuous improvement methods. During the next two decades most Japanese companies adopted a similar system and by 1980 their products dominated the world market. The performance gap between the Japanese and Western manufacturers was highlighted by Womack *et al.* (1990). After surveying about half of the world's car assembly capacity, they claimed that the productivity and quality of the American and European car manufacturers was less than half of that of the average Japanese company. Western plants also carried much higher inventories. According to Womack *et al.* (1990) this performance difference was due to Japanese 'lean' manufacturing method. The principles of the concept are summarised in table 2.1.

| Production (Toyota model) | Product development (Honda model) |
|---|---|
| • Just-in-time small-lot production | • Rapid model replacement |
| • Minimal in-process inventories | • Frequent model-line expansion |
| • Geographical concentration of assembly and parts production | • Overlapping and compressed development phases |
| • Manual demand-pull with kanban cards | • High levels of supplier engineering |
| • Production levelling | • "Heavyweight" project managers |
| • Rapid setup | • Design team and manager continuity |
| • Machinery and line rationalisation | • Strict engineering schedules and work discipline |
| • Work standardisation | • Good communication mechanisms and skills |
| • Foolproof automation devices | • Multi-skilled engineers and design teams |
| • Multi-skilled workers | • Skilful use of computer-aided design tools |
| • High level of subcontracting | • Continuous incremental product improvements |
| • Selective use of automation | |
| • Continuous incremental process improvement | |

**Table 2.1** Principles of lean manufacturing and product development (Cusumano 1994)

The debate over lean production practices dominated the last decade of the twentieth century. Western companies generally regarded the concept as a valuable tool for improving their performance and competitiveness (e.g. Chase and Aquilano 1992, Oliver *et al.* 1994, Spear and Bowen 1999). However, criticism started to emerge when the manufacturers experienced difficulties in adopting the Japanese management style. Reservations were expressed concerning the existence of a unified system (Warnecke and Huser 1995, Muffatto 1999), the transferability of the concept to other countries (Spina *et al.* 1996), the differentiation from the mass production paradigm (Ellegard *et al.* 1992, Willis 1998, Bartezzaghi 1999, Katayama and Bennett 1999), the social and economic implications (Cusumano 1994, Berggren 1994, Katayama and Bennett 1996, Bartezzaghi 1999), and the robustness and viability of the concept to cope with changes in the marketplace (Suda 1989, Katayama and Bennett 1996, Gould 1997, Griffiths *et al.* 2000, McCurry and McIvor 2002). According to Katayama and Bennett (1999) the success of Japanese companies during the 1980's was mainly due to the favourable economic and social conditions that sustained the lean manufacturing strategy of constant market share expansion via cost reduction and increasing product variety. The economic crisis of the 1990's broke this cycle and brought about a more competitive environment where the stable demand required by profitable lean production was more difficult to obtain. On one hand this lead to attempts to modify the concept (e.g. Muffatto 1999, Hines *et al.* 2004) and on the other to calls to develop a new more adaptable paradigm (e.g. Katayama and Bennett 1996, Kidd 2000).

## 2.2 Present and future challenges in manufacturing

The 1990's were characterised by an increasing speed of technological innovations and social, political and economic changes (Kidd 2000). New developments in communication and transportation technologies improved the speed and cost-efficiency of information exchange and the movement of people and goods (Featherston 1999). This facilitated the integration of national economies and dramatically increased international trade. Moreover, the technological advancements made physical locations of management and manufacturing facilities less important and enabled companies to relocate their production operations to countries with lower labour costs (Hughes 1997). Featherston (1999) also pointed out that the general trend at the time towards democratic and liberal economies and global free-trade policies opened up new markets and exposed the already saturated and stagnant domestic markets to competition from abroad. The new competitors introduced product variety to markets previously controlled by few domestic producers. At the same time, the increasing standard of living, growing individualism and the emergence of new social groups diversified the demand and helped to fragment the stable and homogenous mass market that had dominated most of the twentieth century (Featherston 1999). Hughes (1997) summarised the drivers of change within manufacturing as i) ubiquitous availability and distribution of information, ii) accelerating pace of change in technology, iii) rapidly expanding technology access, iv) globalisation of markets and business competition, v) global wage and job skills shift, vi) environmental responsibility and resource limitation, and vii) increasing customer expectations. Clark and Fujimoto (1991), Nagel and Dove (1991), Doll and Vonderembse (1992) and Levary (1992) were some of the first authors to identify the emergence of a new dynamic manufacturing environment. More recently several other authors including Warnock (1996), Bongaerts *et al.* (1997a), Gunasekaran (1998), Featherston (1999), Koste and Malhotra (1999), Sharifi and Zhang (1999), Hitt (2000), Kidd (2000), St. John *et al.* (2001) and Manzini *et al.* (2004) have made similar observations. Figure 2.1 illustrates the main forces in the post-2000 marketplace.

The effects of these emerging and continuing trends are more demanding customers, greater competitive intensity, and increased complexity in production technology and coordination (St. John *et al.* 2001). Today's highly informed, sophisticated and

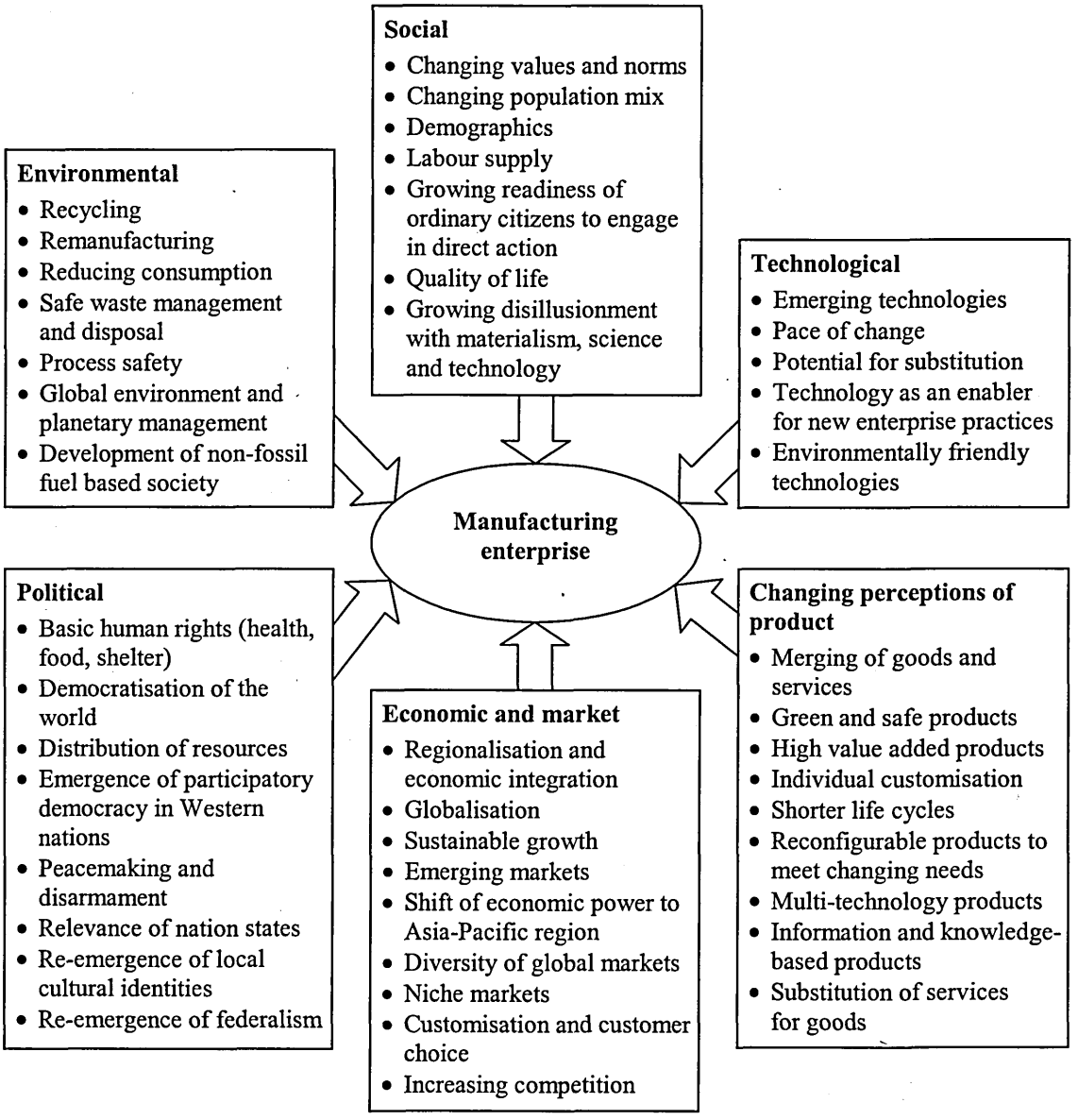



**Figure 2.1** Main drivers of change in manufacturing enterprises (Kidd 2000)

demanding consumers expect companies to continuously deliver lower prices, better quality, a wider range of products, and a faster and better service (Warnock 1996). The outcome is a growing demand for customised products, faster delivery time and time to market, and rapid changes in order quantity and specification (Sharifi and Zhang 1999). With companies facing aggressive competition on a global scale, the ability to rapidly identify and satisfy the specific and individual needs of customers is becoming critical to competition in many industries (Doll and Vonderembse 1992). Consequently, the lines of competition have moved from the ability to supply the demand to product price, quality, delivery time, after-sales service, and customer choice i.e. customer satisfaction (Jin-Hai *et al.* 2003). Furthermore, operational capabilities to provide flexibility, speed

and responsiveness to customers are quickly replacing the traditional product characteristics of quality, price and design as the order-winning criteria (Willis 1998). To comply with these new requirements, manufacturing systems need to be able to cope efficiently with large fluctuations in product demands, increasing product variety, and shortening product life cycles (Koren *et al.* 1998, Sharifi and Zhang 1999). McFarlane and Bussmann (2000) summarised the challenges faced by manufacturers as more complex products, faster changing products, faster introduction of products, a volatile output, and reduced investment. Thus, companies now compete in a climate of uncertainty, unpredictability and highly turbulent market conditions (Esmail and Saggu 1996) characterised by continuous change (Koste and Malhotra 1999). Sharifi and Zhang (1999) pointed out that the market changes are occurring faster and more unexpectedly than ever before. Moreover, Davis (1995) identified the ability to adapt to and manage change as the most common problem facing organisations today, while Small and Downey (1996) noted that turbulent times and uncertainty are the main causes for failure in the manufacturing industry.

To stay competitive in this new dynamic and uncertain environment, manufacturing companies need to be able to respond quickly and efficiently to unexpected changes and disturbances (Ramasesh *et al.* 2001). Traditional manufacturing systems based on mass production principles and designed for long-term, high-volume production of only a few standardised products are often unable to cope under dynamically changing circumstances, e.g. frequent changes in process requirements and production orders (Koren *et al.* 1998, McCarthy and Tsinopoulos 2003). This is due to their structural rigidity, deterministic approach to decision making in a stochastic environment, hierarchical allocation of competencies, and insufficient communication and exploitation of expertise (Sluga and Butala 2001). While in the past the ability to cost-effectively produce a single product was enough for market success today it requires flexibility, agility and versatility i.e. ability to handle continuous improvements and change (Jin-Hai *et al.* 2003). Hence, there is a growing demand for more flexible, autonomous, adaptable, reconfigurable and reliable production solutions (Ryu and Jung 2003, Simsek and Albayrak 2003). According to Benjaafar and Sheikhzadeh (2000), today's manufacturing systems should at foremost be designed to accommodate the simultaneous production of several part types in varying quantities, and rapid and

smooth product line shifts without major retooling, resource reconfiguration or equipment change. To handle the complexity and dynamism in the manufacturing environment, several new paradigms have recently been proposed and studied in the academic literature. Wyns (1999) noted that the novel concepts such as agile (e.g. Gunasekaran 1998, Sharifi and Zhang 2001), holonic (e.g. Van Brussel *et al.* 1998, McFarlane and Bussmann 2000), fractal (e.g. Warnecke 1993, Venkatadri *et al.* 1997) and biological (e.g. Ueda *et al.* 1997b, Ueda *et al.* 2001a) manufacturing have been especially designed to meet the requirements of an unstable and unpredictable marketplace. They also offer more flexible, responsive and adaptable structures and processes than the traditional manufacturing approaches (Tharumarajah *et al.* 1996, Sousa *et al.* 1999).

## 2.3 Reference architectures

The first solution to a sufficiently difficult task can often be adopted as template to resolve future problems of the same kind. A specific architecture or model in any domain can be referenced to by others working in the same domain for comparison and to emphasise shared characteristics (Wyns 1999). The copies and adaptations of the architecture normally differ in some respects but adhere to a set of coherent fundamental design principles (Kidd 2000). Hence, the specific reference model can set a common language in the domain if it is used by a large amount of people working in the field. This simplifies communication and facilitates a conceptual understanding through direct comparison with it (Williams *et al.* 1994).

In most domains no single specific architecture could satisfy all requirements of any arbitrary design problem. Specific factors attributed to a variety of case related characteristics need to be taken into account to define a suitable system architecture (Wyns 1999). Kidd (2000) refers to these parameters as case sensitive complexity factors caused by unique characteristics of each business sector and company which in turn cause system architectures to differ. A reference architecture can provide a framework where some of its features are universally valid and present useful solutions for a large number of design problems (Wyns 1999). A system architecture would then be a composite of well-known elements from a reference architecture and necessary adaptations to cater for the specific requirements (Wyns 1999), as illustrated in figure

2.2. Where specific system architectures tend to be fairly dissimilar due to case sensitive factors, a reference architecture would benefit from ignoring any features that are strongly dependent on local factors to reduce system complexity by removing these. In such a scenario a reference architecture would be more useful as theoretical model with a high level of abstraction. It would need to be generic and may not need to exist as a proven and functional system to serve as a template. Williams and Vosniakos (1996) recommend generic and reusable reference models to support companies as foundation for design or comparison. Burkel (1991) describes reference architecture as a framework to guide the project during design and implementation by the means of a structured methodology, the formalisation of operations and the support tools.

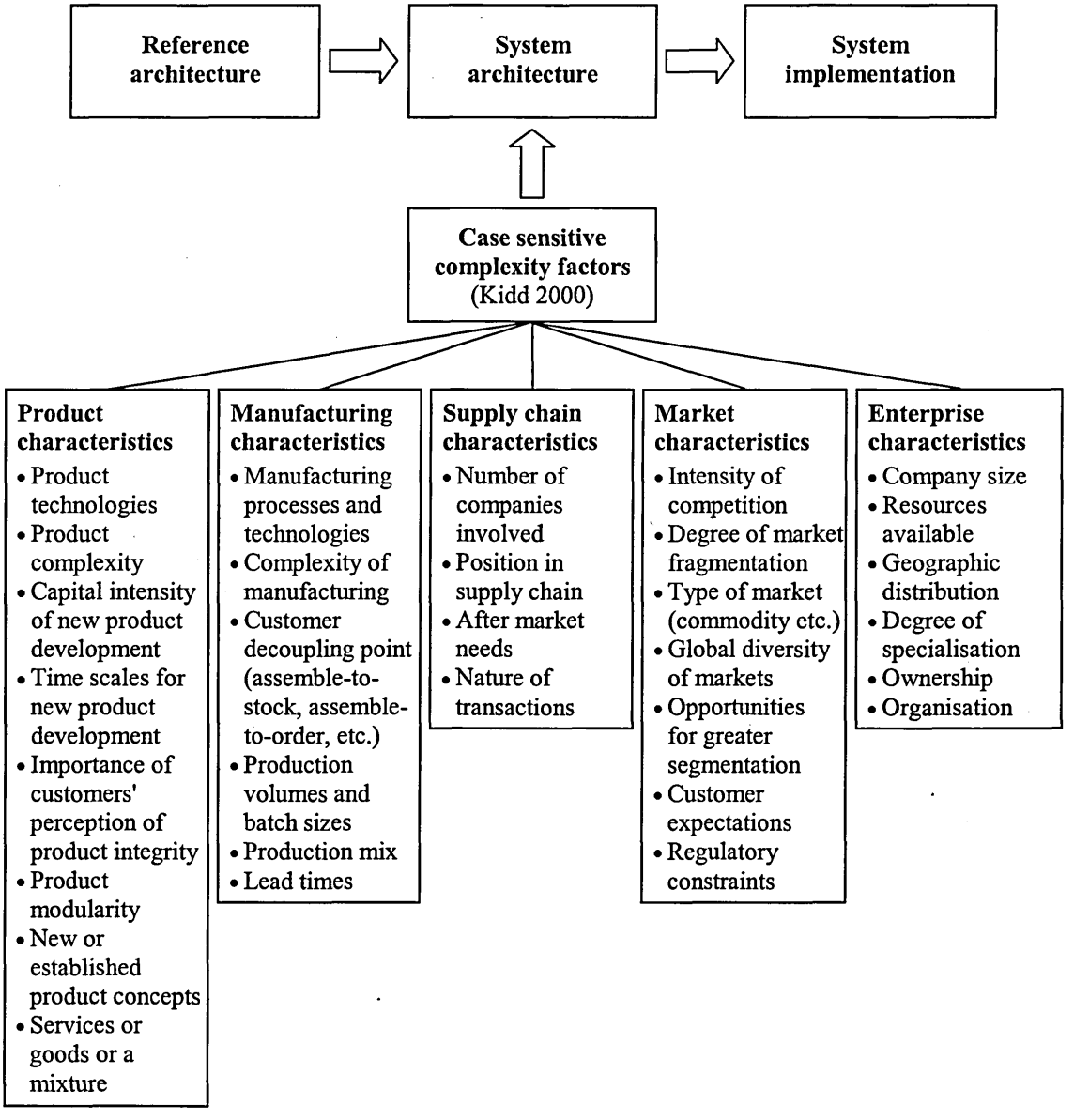



**Figure 2.2** Relationship between reference architecture and system architecture

With regard to reference models, Buchel *et al.* (1984) stress the importance of conceptual architectures to the design process of manufacturing planning and control. The emergent manufacturing methodologies may serve as guiding design principles rather than exact blueprint for possible implementation in specific system architectures. Perry and Wolf (1992) recognise the reference architecture as architectural style that describes well-known elements and their functions, interactions, dependencies and constraints, which the designer can use in ways that are appropriate to relevant case sensitive factors. On this basis a generic and theoretical reference architecture can be developed as a design guide to manufacturing systems (Chalmeta *et al.* 2001). Elements and functions of new and emerging manufacturing paradigms can be incorporated and a common language and taxonomy can be developed to emphasise commonality and shared features among various theoretical manufacturing methodologies and any specific system architectures.

## 2.4 Traditional manufacturing systems

Manufacturing is the process of converting raw material into finished products (Kalpakjian and Schmid 2001). It is generally a complex activity, which in addition to the direct production functions include a variety of operations such as product design, process development, plant design, capacity management, product distribution, plant scheduling, quality control, workforce organisation, equipment maintenance, strategic planning, supply chain management, etc. (Hopp and Spearman 2000). These operations form the managerial system, which support and control the manufacturing activities and ensure that the system achieves its principle objectives, i.e. to produce the specified products on schedule and at a minimum cost (Chase and Aquilano 1992). Simultaneously, for a company to be successful in the long term, the customer service objectives have to be satisfied with efficient operations, i.e. efficient use of resources (Wild 1993). However, as it is not usually possible to maximise performance on all aspects of both objectives, companies must balance and prioritise them to achieve a satisfactory performance and to distinguish themselves from others in the marketplace (Wild 1993). The chosen strategy influences the design, planning, operation and control of the manufacturing system resulting in a unique production scenario for each company (Wild 1993).

Traditional manufacturing systems refer to the production concepts developed immediately after the Second World War to satisfy a high demand for low-cost standardised products (Doll and Vonderembse 1992). At that time, large extensively automated factories with complex organisational structures were used to obtain the economies of scale associated with mass production (Jin-Hai *et al.* 2003). The aim was to achieve low unit cost by spreading fixed costs over the largest possible volume of output (Willis 1998). Although the demand has since diversified, mass production based systems are still dominant manufacturing concepts (Gunasekaran 1998). Traditionally, manufacturing systems have been categorised in terms of the manner in which materials move through the plant, i.e. process structures, or in terms of the physical arrangement of manufacturing resources, i.e. facility layouts. Both of these classifications focus on the movement and flow of materials and the volume and variety capabilities of the systems.

### 2.4.1 Process structures

The term process structure refers to the manner and nature of the flow of materials through a plant, i.e. the routes, flow rate and throughput time of the products (Wild 1993). In general, this flow can be continuous, repetitive or intermittent. Krajewski and Ritzman (2005) classified the manufacturing systems by process type into five categories, namely project, job, batch, line and continuous processes. Figure 2.3 illustrates the different process structures in relation to product types, i.e. the volume-variety characteristics of the product demand. The figure indicates that increasing volume and decreasing product variety correspond to lower unit costs, more specialised equipment and standardised material flows (Chase and Aquilano 1992).

Project processes are used to manufacture expensive and highly customised or one-of-a-kind products, such as construction, airplanes and ships, in low volume and high variety (Chase and Aquilano 1992). Job processes (job shops) provide maximum flexibility for the production of low-volume highly customised items using shared resources and a high variety of routings through the plant (Chase and Aquilano 1992). In batch production products are processed in small lots according to customer specification using disconnected flow lines with a limited number of identifiable routings (Hopp and Spearman 2000). In mass or line production a narrow variety of items with similar work contents are produced in large lots over a significant period of time using connected

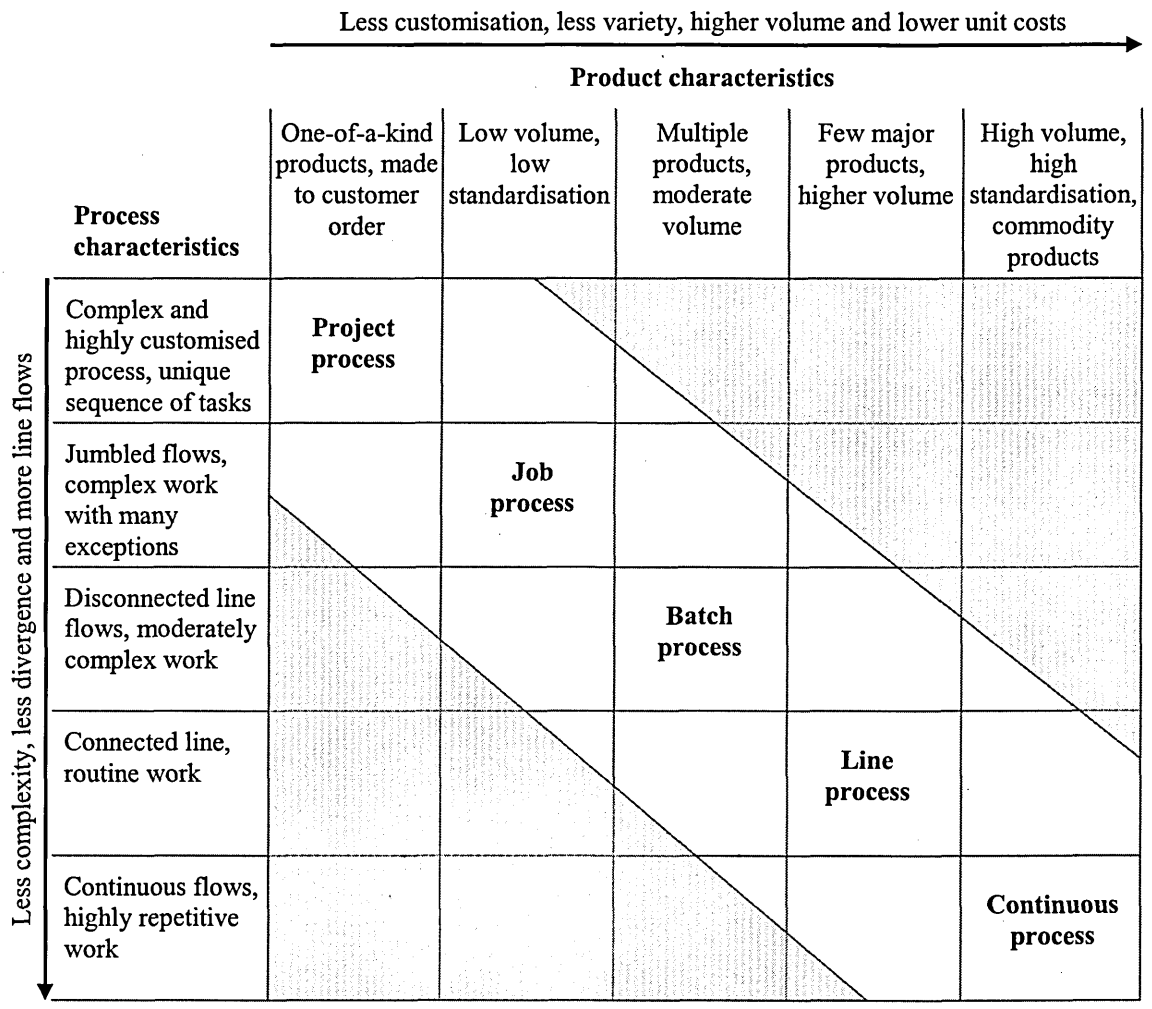



**Figure 2.3** Product-process matrix (Krajewski and Ritzman 2005)

flow lines (Wild 1993). Finally, continuous processes deal with non-discrete products that automatically flow down a fixed route for an extensive period of time (Hopp and Spearman 2000). These processes are typically found in process industries.

### 2.4.2 Layout types

The study of facility layout relates to the arrangement of physical production resources within a productive facility (Chase and Aquilano 1992). Its general objectives are to minimise the physical movement and handling of materials, to maximise the capacity utilisation (Wild 1993) and to ensure a smooth work flow (Chase and Aquilano 1992) in accordance with the system objectives. Four basic types of shop floor configurations can be identified, namely fixed position, process, cellular and product layouts (Chase and Aquilano 1992). Figure 2.4 illustrates the arrangement of resources in each layout type, while figure 2.5 shows the volume-variety characteristics of the different process

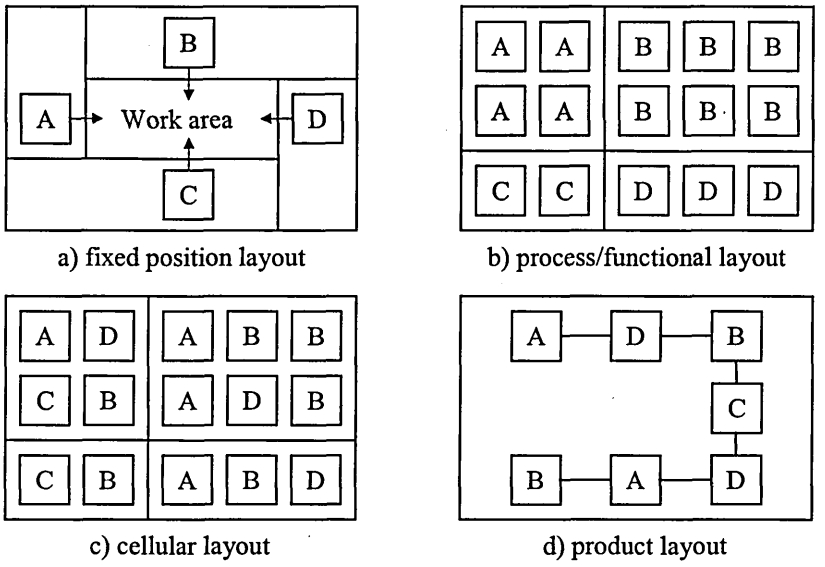



**Figure 2.4** Arrangement of facilities in different layout types

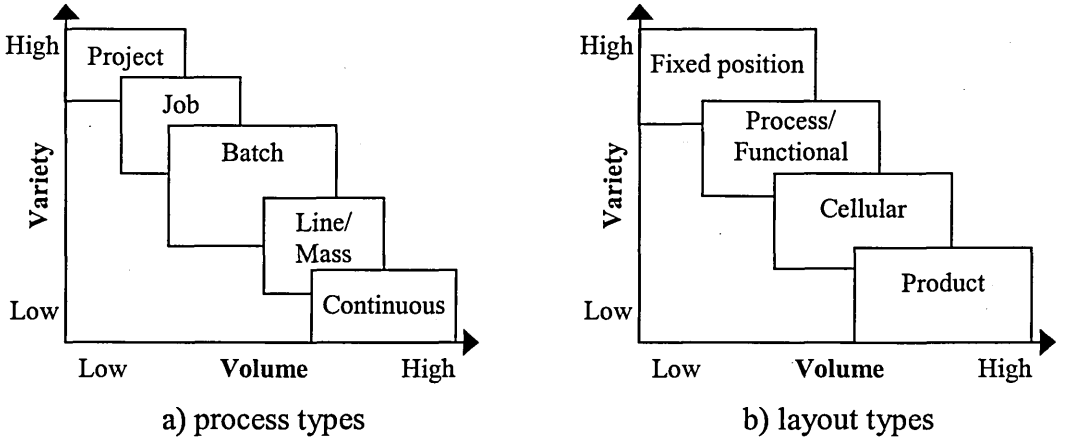



**Figure 2.5** Volume-variety characteristics of different process and layout types

structures and layout types. The latter figure suggests that each layout type is suitable for a particular form of manufacturing, depending on the volume and the variety of products involved (Wild 1993).

In a fixed position layout the product remains at one location while manufacturing equipment moves to the product (Chase and Aquilano 1992). These configurations are mainly used in civil engineering projects and the manufacture of large items such as ships and aircraft (Wild 1993). With a process or functional layout all operations of a similar nature are grouped together in the same department (Wild 1993), as illustrated in

figure 2.4b. Since the products in this layout travel from department to department for processing, the functional areas are arranged on the shop floor according to the main flow patterns in order to minimise routing distances (Chase and Aquilano 1992). These functional layouts enable a high degree of production flexibility, but suffer from high levels of work-in-progress, long throughput times, and excessive material handling costs, and are therefore best suited for low volume-high variety manufacture (Wild 1993).

To achieve a cellular or group layout functionally dissimilar machines are grouped together into cells and dedicated for the processing of a family of parts, i.e. parts with similar processing requirements (Farrington and Nazemetz 1998). The objective is to streamline material flows and to minimise the setups in the batch manufacture of products (Co and Araar 1988). Cellular layouts are considered a feasible option when distinctive part families exist, multiple copies of each type of machine are available, and if the machines are easily movable (Chase and Aquilano 1992). The procedure used to specify part families and cell contents is referred to as group technology (Wild 1993). It consists of three phases: i) development of a classification and coding system for items, ii) grouping parts into families with similar processing requirements and routings, and iii) creating the physical layout by positioning machines into cells and by positioning cells relative to each other (Chase and Aquilano 1992). The benefits of these cellular configurations are summarised by Sarker and Li (2001) as reduced setup times, improved throughput times, reduced work-in-process inventories, reduced material handling costs, and improved material flow and machine utilisation. However, due to their fixed structure and limited routing flexibility, group layouts are often unable to respond to changes in demand patterns (Kochikar and Narendran 1998) without expensive and time-consuming physical reconfiguration (Sarker and Li 2001). The layout design problem in cellular manufacturing has been widely studied in the academic literature, e.g. Agarwal and Sarkis (1998), Akright and Kroll (1998), Billo (1998), Wang *et al.* (2001) and Molleman *et al.* (2002).

The last type of product layouts envisions facilities to be arranged in line with the processing sequence required by the product and linked by a material handling device (Chase and Aquilano 1992). These configurations are generally appropriate for high

volume-low variety manufacturing of complex standardised products, i.e. mass production (Wild 1993). Traditionally, the flow lines were designed to handle only one type of product or model (Bukchin 1998). The goal was to minimise the production costs by maximising the volume (Wild 1993). However, the changing customer demand has increased the variety of models of each product that needs to be assembled (Duplaga and Bragg 1998). This has resulted in the development of multi-model and mixed-model flow lines to coincide with the single-model lines. The cost-effective production of more than one product on a single line has been made feasible by flexible manufacturing systems (Bard *et al.* 1992). However, the improvements in dealing with product variety are limited to manufacture of products with similar work content (Wild 1993). An important drawback of product layouts is that a single machine breakdown can stop whole line (Wild 1993). To improve system reliability, Koren et al. (1998) proposed the use of parallel and hybrid configurations instead of the standard serial layout. In general, product layouts are relatively inflexible, but can in a stable high demand environment provide minimum work-in-progress, low throughput times, minimum material handling costs, and high machine utilisation, i.e. low unit costs (Wild 1993).

### 2.4.3 Production planning and control

The production planning and control (PPC) function and its associated systems are responsible for the planning and control of manufacturing activities with the aim of satisfying the production requirements as effectively as possible (Bonney 2000). It is a hierarchical procedure where different levels operate in different time scales, i.e. with a long-term, medium-term or short-term planning horizon (Chase and Aquilano 1992). Although the production planning and control systems vary among different companies and production scenarios (i.e. the range of products to be produced, production volume requirements, product due-date requirements, the production system capacity, the management and production strategies adopted, and the part transfer approach to be used), the basic structure and components of the system remain the same (England 2004). This basic framework is illustrated in figure 2.6.

Traditionally, in a mass production environment products are produced according to the demand forecast (England 2004). This production strategy, known as make-to-stock, focuses on ensuring the product availability by carefully managing the stock levels

(Wild 1993). However, the growing demand for customised products has resulted in a shift towards more customer focused approaches, namely make-to-order, where products are produced only after an order request has been received from a customer (England 2004). Although this strategy reduces the storage costs and increases flexibility, the time from order to delivery is longer (England 2004). Hence, it is only suitable for manufacture where customers are willing to wait longer for the delivery. The lack of information on the short-term planning horizon makes inventory planning and production scheduling difficult, especially where the system was designed for stable make-to-stock environment (England 2004). Thus, it is likely that unpredictable demand patterns result in high variations in system utilisation (Rahimifard *et al.* 1999). Yeh (2000) and England (2004) noted that the make-to-order production strategies can yield great benefits in supporting modern order requirements; however better planning and control techniques are required. In addition to the make-to-stock and make-to-order strategies, several other approaches that offer varying levels of customer service have recently been proposed in the academic literature, e.g. assemble-to-order (Inman and Schmeling 2003) and engineer-to-order (Little and Rollins 2000).

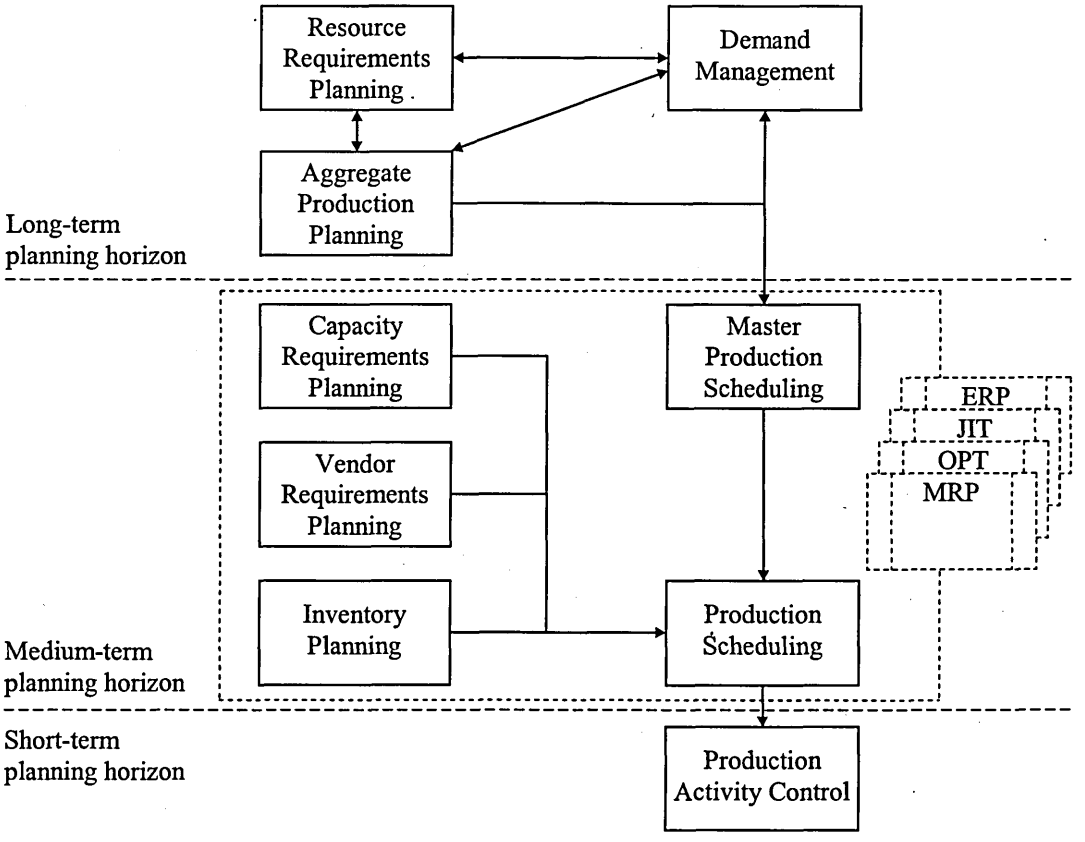



**Figure 2.6** Basic production planning and control framework (England 2004)

Demand management, resource requirement planning and aggregate production planning are based on long-term decisions performed at the highest level of the production planning and control hierarchy, as illustrated in figure 2.6. Planning requires an estimated forecast of the future product demand, calculation of the level of capacity required to meet this demand in a cost-efficient way, and a specification of the optimal combination of production rates, work force levels, and inventory holdings to meet expected fluctuations in the demand (Wild 1993). Together these activities help to formulate the master production schedule that identifies over a medium time period (i.e. weeks) what will be produced and when (Chase and Aquilano 1992). The function of production scheduling then divides the master production schedule up into a short-term (i.e. days and hours) schedule by considering the capacity requirements, vendor requirements and inventory plans (England 2004). The production schedule includes information about when each product is to be processed on which machine and in which order (Hopp and Spearman 2000). Its main objective is to generate a feasible work plan for the shop floor operations that meets the order due dates and optimises resource utilisation (England 2004). Traditionally, scheduling has been performed prior to production using rigid and static plans (England 2004). However, the recent changes in customer expectations have created a demand for more dynamic and flexible scheduling approaches that enable frequent re-scheduling based on the latest system status i.e. changes in production orders and resource availability (e.g. Alvarez and Diaz 2004, Babiceanu *et al.* 2005). The execution of production schedules on the shop floor on a day-to-day basis is the responsibility of production activity control (England 2004). Its major functions are shop planning (to ensure resource availability), order dispatching, lot control, managing changes to shop orders (e.g. rework), and providing feedback of operational performance (Chase and Aquilano 1992). Brennan and Norrie (2001) and Maione and Naso (2001) noted that the traditional shop floor control systems lack the flexibility required by today's dynamic environment.

Materials requirements planning (MRP), manufacturing resources planning (MRP2), enterprise resource planning (ERP), just-in-time (JIT) and optimised production technology (OPT) are production planning and control systems that link scheduling and inventory systems together enabling companies to manage their inventories and resources more efficiently in a stable and predictable manufacturing environment

(England 2004). The MRP system uses the master production schedule, the bill-of-materials and inventory data to identify detailed materials requirements and production plans to satisfy demand (Wild 1993). As this plan is created without considering the capacity constraints of the production system, its feasibility needs to be evaluated. This is the activity of capacity requirements planning. MRP2 and ERP are extended versions of MRP and are known as 'push' systems, while the OPT and JIT are 'pull' systems (England 2004). These traditional production planning and control systems have received extensive criticism in recent years due to their rigid hierarchical structures and downstream information flows that restrict system reconfiguration, reliability and expansion i.e. reactivity to disturbances (e.g. Gou *et al.* 1998, Giebels *et al.* 1999, Bongaerts *et al.* 2000, Wang 2001). In addition, Hopp and Spearman (2000) highlighted problems with long lead times and capabilities to implement changes, i.e. small changes in the demand can cause major changes in the shop floor, hence changes are avoided.

## 2.5 Emerging manufacturing paradigms

The inability of the traditional manufacturing systems to adapt to changing market conditions have been widely recognised (e.g. Gunasekaran 1998, Katayama and Bennett 1999, Sharifi and Zhang 1999). As a result, several new production theories that go beyond the conventional models of manufacturing have recently been proposed and studied in the academic literature, e.g. agile (e.g. Gunasekaran 1998, Sharifi and Zhang 2001), holonic (e.g. Van Brussel *et al.* 1998, McFarlane and Bussmann 2000), fractal (e.g. Warnecke 1993, Venkatadri *et al.* 1997) and biological (e.g. Ueda *et al.* 1997b, Ueda *et al.* 2001a) manufacturing. The fundamental problem these emerging paradigms face is how to achieve the cost-efficiency of mass production systems in a rapidly changing high variety and low volume production environment. Figure 2.7 summarises some of the most promising and best known new concepts.

### 2.5.1 Agile manufacturing

Agile manufacturing was one of the first new paradigms to emerge. It was proposed by Nagel and Dove (1991) as a tool for restoring the competitiveness of the American industry. The researchers defined agile manufacturing as "the ability of an organisation to thrive in the competitive environment of continuous and unanticipated change and to respond quickly to rapidly changing markets driven by customer based valuing of

products and services". Thus, agility comprises two main factors: i) responding to changes (anticipated or unexpected) in a proper way and in due time, and ii) exploiting changes and taking advantage of them as opportunities (Sharifi and Zhang 2001). The concept builds on four underlying principles, namely delivering value to customers, being ready for change, valuing human knowledge and skills, and forming virtual partnerships (Goldman *et al.* 1995), as illustrated in figure 2.8. Yusuf *et al.* (1999) described agility as "the successful exploration of competitive bases, i.e. speed, flexibility, innovation, reactivity, quality and profitability, through the integration of reconfigurable resources and best practices in a knowledge-rich environment to provide customer-driven products and services in a fast changing market environment". Sanchez and Nagi (2001) noted that "agility is characterised by cooperativeness and synergism, by a strategic vision that enables thriving in face of continuous and unpredictable

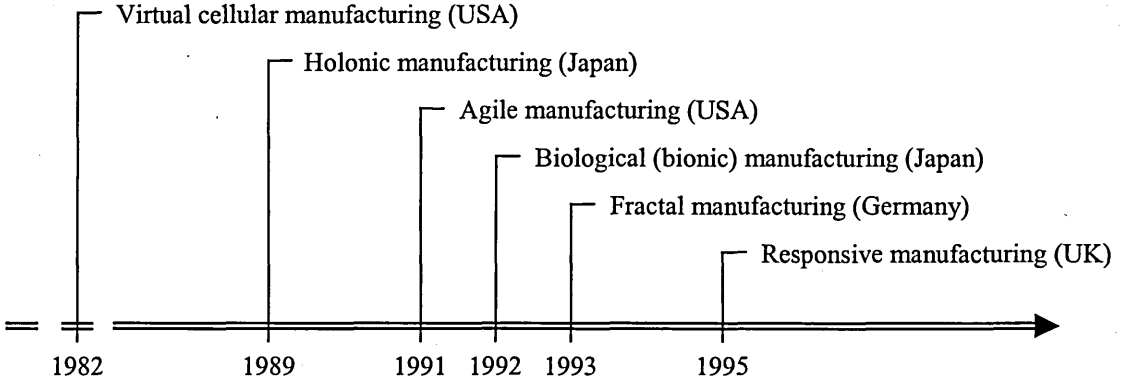



**Figure 2.7** Timescale for emerging manufacturing concepts

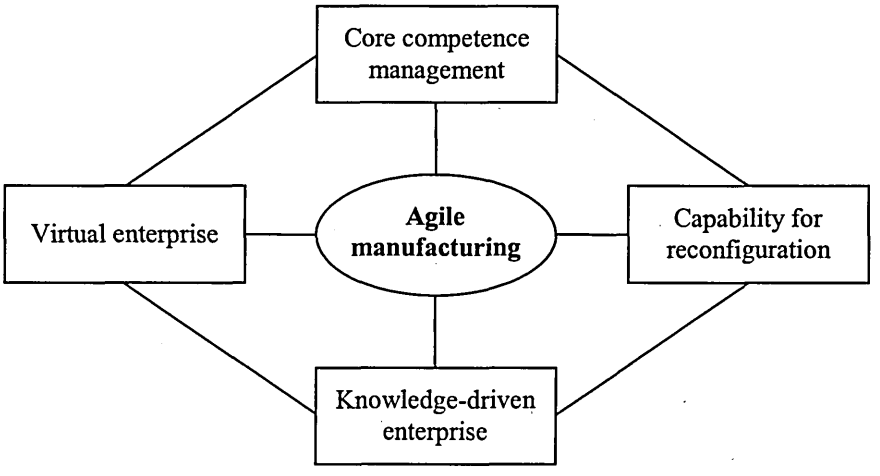



**Figure 2.8** Core concepts of agile manufacturing (Yusuf *et al.* 1999)

change, by the responsive creation and delivery of customer-valued, high quality and mass customised goods and services, by nimble organisation structures of a knowledgeable and empowered workforce, and facilitated by an information infrastructure that links constituent partners in a unified electronic network". Hence, agility is: i) a response to change and uncertainty, ii) the building of core competencies, iii) the supply of highly customised products, iv) the synthesis of diverse technologies, and v) intra-enterprise and inter-enterprise integration (Jin-Hai *et al.* 2003). In effect, agility is the capability to respond flexibly and respond speedily (Conboy and Fitzgerald 2004).

To realise the strategic and operational benefits of agile manufacturing Meredith and Francis (2000) proposed an agile manufacturing reference model, illustrated in figure 2.9, which provides an integrated definition of the components of agility. They also identified the following attributes of an agile manufacturing facility i) it produces products to order, ii) it meets the customer's specific needs, iii) it achieves a speed and flexibility in its functioning that equals the speed and flexibility of its technologies, iv) it mobilises and manages knowledge intelligently, v) it adopts new ways of working,

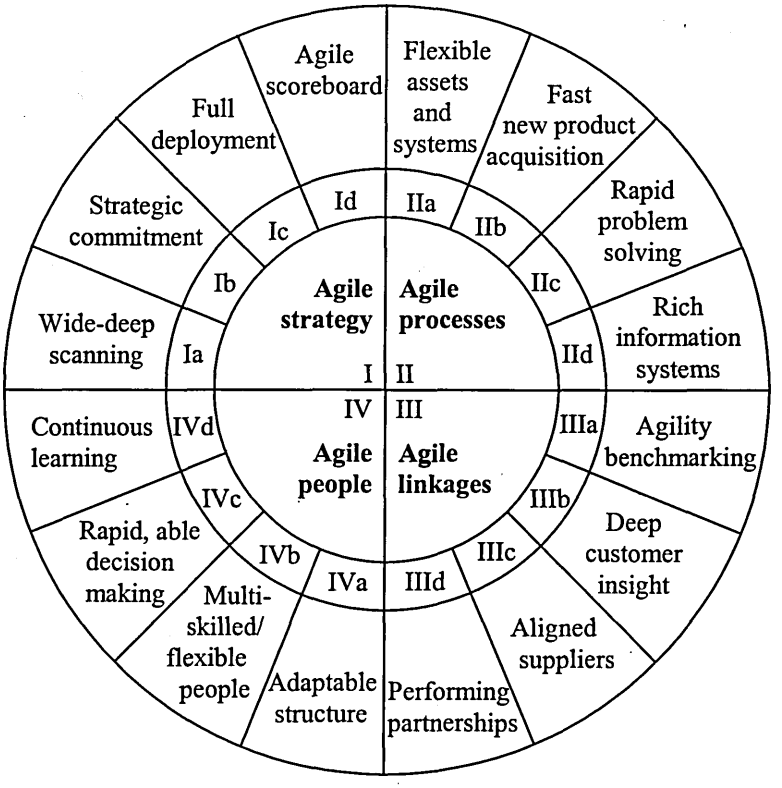



**Figure 2.9** Agile manufacturing reference model (Meredith and Francis 2000)

e.g. team working, and vi) it creates virtual projects and temporary organisations to add capabilities when needed. Yusuf *et al.* (1999) recognised similar attributes in other decision domains. Their summary of agile practices is presented in table 2.2. Likewise, Willis (1998) noted that the operational competitive requirements of agile manufacturing, i.e. speed, flexibility, and responsiveness to customer, can be achieved by streamlining the operational functions of a company, namely customer order and delivery process, product development, production process, and supplier network. In essence, this involves i) horizontal and parallel business processes, ii) close relationships between management and the customer, iii) functional jobs in a flexible focused factory environment, iv) fewer management levels, v) the rapid response to new market opportunities and threats, vi) uniformity with flexibility, vii) fast development, production, and release of products, viii) the ability to rapidly modify products and change output volumes, and ix) fast delivery of customised products (Willis 1998). The implementation of a system using agile manufacturing requires capabilities for virtual enterprise formation, physically distributed teams and manufacturing, rapid partnership formation, concurrent engineering, integrated product/production/business information

| | |
|---|---|
| **Integration attributes:** | **Competence attributes:** |
| Concurrent execution of activities | Multi-venturing capabilities |
| Enterprise integration | Developed business practice difficult to copy |
| Information accessible to employees | |
| **Team building attributes:** | **Technology attributes:** |
| Empowered individuals working in teams | Technology awareness |
| Cross functional teams | Leadership in the use of current technology |
| Teams across company borders | Skill and knowledge enhancing technologies |
| Decentralised decision making | Flexible production technology |
| **Welfare attributes:** | **Change attributes:** |
| Employee satisfaction | Continuous improvement |
| | Culture of change |
| **Partnership attributes:** | **Market attributes:** |
| Rapid partnership formation | New product introduction |
| Strategic relationship with customers | Customer-driven innovations |
| Close relationship with suppliers | Customer satisfaction |
| Trust-based relationship with customers/suppliers | Response to changing market requirements |
| **Education attributes:** | **Quality attributes:** |
| Learning organisation | Quality over product life |
| Multi-skilled and flexible people | Products with substantial value-addition |
| Workforce skill upgrade | First-time right design |
| Continuous training and development | Short development cycle times |

**Table 2.2** Attributes of an agile organisation (Yusuf *et al.* 1999)

systems, rapid prototyping, and electronic commerce so that the physically distributed companies can be integrated and managed effectively (Gunasekaran 1998).

Several other authors have studied and further developed the concept of agility and its applications. Moskal (1995) and DeVor *et al.* (1997) found that real industry projects which adopted agility did primarily focus on business practices, processes and technology. Rigby *et al.* (2000) studied the inter-organisational relations in agile networks. Esmail and Saggu (1996), Gunasekaran (1998), Willis (1998), Sharifi and Zhang (1999, 2001), Yusuf *et al.* (1999) Meredith and Francis (2000) Christian *et al.* (2001) and Jin-Hai *et al.* (2003) developed frameworks and methodologies for achieving agility in manufacturing organisations. Figure 2.10 illustrates one of the suggested methodologies. Jackson and Johansson (2003) and Arteta and Giachetti (2004) proposed methodologies for measuring a company's level of agility. Ramasesh *et al.* (2001) proposed a quantitative analysis framework and a simulation methodology to explore the value of agility in financial terms. Wadhwa and Rao (2003) investigated the relationship between flexibility and agility and concluded that flexibility enables agility. Yusuf *et al.* (2003) studied the flexibility of agile organisations and suggested the use of virtual cells as the means of improving volume flexibility. Finally, He *et al.* (2001) developed a heuristic algorithm for solving scheduling problems in agile manufacturing systems, while Quinn *et al.* (1997) examined the design of agile work cells.

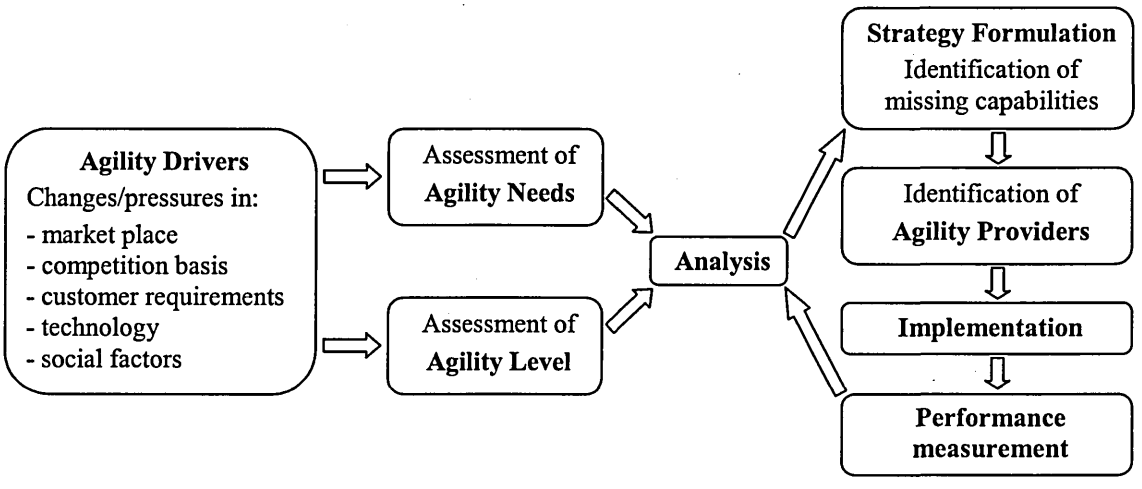



**Figure 2.10** Methodology for agility (Sharifi and Zhang 2001)

### 2.5.2 Holonic manufacturing

The concept of holonic manufacturing was proposed by Suda (1989, 1990). It is based on the work by Koestler (1967) who first suggested the term 'holon' to describe a basic structural unit in social systems and living organisms. He noted that these systems evolve to satisfy changing needs by creating stable and self-reliant intermediate forms in dynamic hierarchical structures. The word 'holon' originates from Greek word holos, which means whole; the suffix 'on' refers to a particle or part (Tharumarajah *et al.* 1996). Hence, a holon is simultaneously a whole (e.g. a machine) and a part of the whole (e.g. a manufacturing system) and has both autonomous and cooperative characteristics (Tharumarajah *et al.* 1996), as illustrated in figure 2.11. The autonomy of the holons is based on their operational features, individual goals and the ability to define their own tasks and execution plans. However, holons cooperate with their lateral partners to combine their competencies and to achieve both, individual and system goals (Sousa *et al.* 1999). The performance of holons is defined by fixed rules that determine their static structural and functional configurations and flexible strategies that define the holons' authorised activities in accordance with the changes in the environment (Tharumarajah *et al.* 1996). Although the holons have a degree of independence that enables them to survive disturbances, they are still subject to control from higher authorities, which ensures that they provide accurate functionality for the bigger whole (Bongaerts *et al.* 1997a). When holons cooperate to achieve a goal or objective, they form a holarchy i.e. a system of holons that enables the construction of complex systems that are efficient in the use of resources, highly resilient to external and internal disturbances and adaptable to changes (Akturk and Turkcan 2000). Sousa *et al.* (1999) summarised the properties of holons as i) autonomous and cooperative nature, ii) to

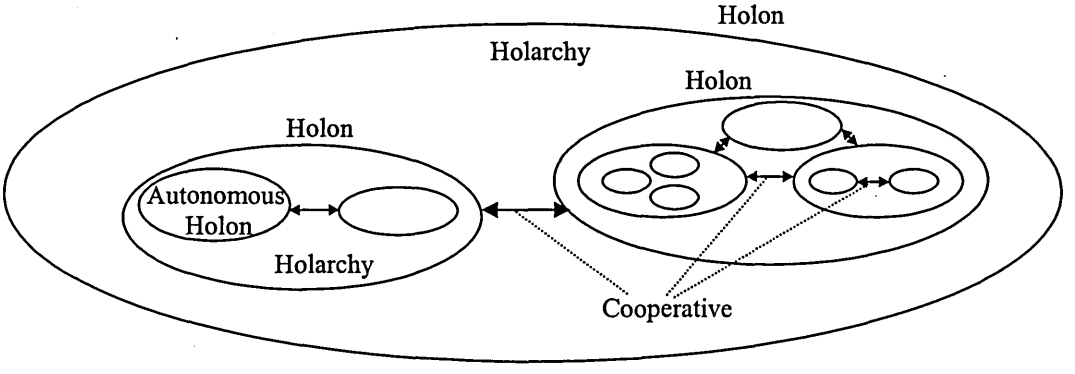



**Figure 2.11** Holonic system (Tharumarajah *et al.* 1998)

hold information about themselves and the environment, iii) to be composed of other holons to form a holarchy, iv) to dynamically belong to multiple holarchies, and v) fixed rules and directives.

For manufacturing applications a holon can be defined as "an autonomous and cooperative building block of a manufacturing system for transforming, transporting, and storing physical and information objects" (McFarlane and Bussmann 2000). Figure 2.12 illustrates the general architecture of a sample holon that consists of a control part and an optional physical processing part. A combination of different holons can support the entire production operation, as holons i) support all production and control functions required to complete production tasks, and ii) manage the underlying equipment and systems (McFarlane and Bussmann 2000). Table 2.3 summarises the features and benefits of holonic manufacturing systems. Valckenaers *et al.* (1998), Van Brussel *et al.* (1998) and Wyns (1999) proposed a reference architecture for holonic manufacturing systems to realize the benefits of holons in manufacturing, namely the stability in the face of disturbances, adaptability and flexibility in the face of change, and the efficient use of available resources. The architecture consists of three types of basic holons, namely order holons, product holons, and resource holons, as illustrated in figure 2.13.

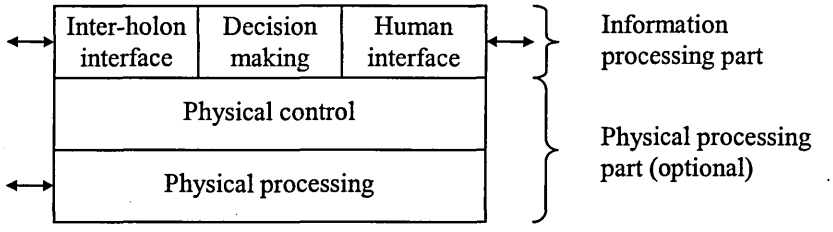



**Figure 2.12** General architecture of a holon (Bussmann 1998)

| Features of holonic manufacturing | Benefits of holonic manufacturing |
|---|---|
| • Bottom up development<br>• Autonomous operation of subordinate holons<br>• Distributed database containing goals for, and state of, all holons updated in real time<br>• Communication network to allow information exchange with neighbouring and remote holons | • Drastic reduction of cost of changes<br>• Reduction of lead times<br>• Better use of human skills and thus better system performance and higher job satisfaction<br>• Higher variability of products<br>• Greater ability to recover automatically from unplanned production stops<br>• Re-usability of automation equipment |

**Table 2.3** Features and benefits of holonic manufacturing systems (Höpf 1994)

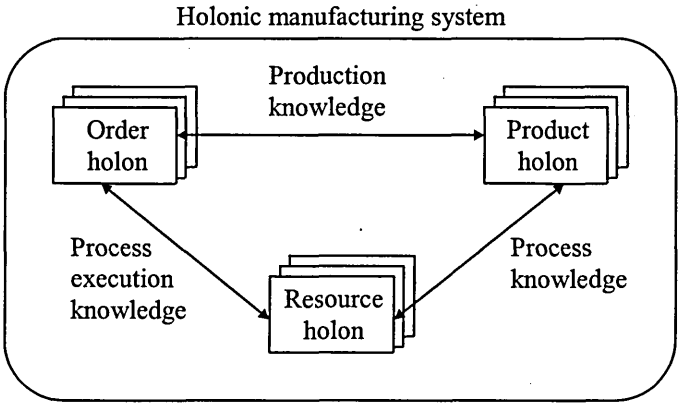



**Figure 2.13** Basic structure of a holonic manufacturing system (van Brussel *et al.* 1998)

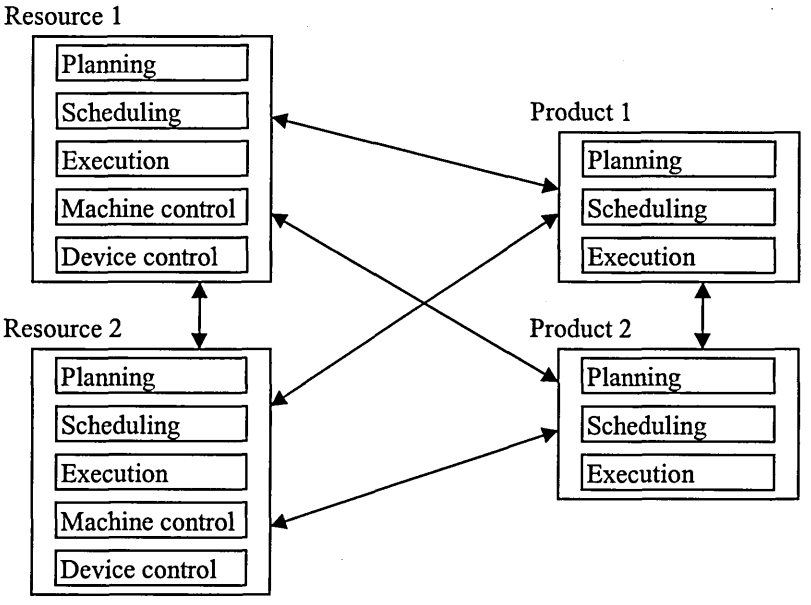



**Figure 2.14** Holonic control mechanism (McFarlane and Bussmann 2000)

Valckenaers *et al.* (1998) explained that this structure reflects the three major concerns in manufacturing i.e. resource aspects, product and process related technological aspects, and logistical concerns. In addition, several other reference architectures for holonic control have been illustrated in the academic literature. Balasubramanian *et al.* (2000, 2001) described an event-driven control architecture for reconfigurable holonic systems. Wang (2001) proposed holonic reference architecture based on a design-to-control concept. Arai *et al.* (2001) studied the architecture of a holonic assembly system. Bongaerts *et al.* (1997b) presented an architecture for reactive scheduling that enables concurrent scheduling and schedule execution in a holonic manufacturing environment, while Sousa and Ramos (1998, 1999) suggested a holon-based negotiation

protocol for dynamic scheduling. Similarly, Markus *et al.* (1996), Gou *et al.* (1998) and Toh *et al.* (1999) proposed methodologies for scheduling in a holonic environment. Fischer (1999), Rahimifard *et al.* (1999) and Wullink *et al.* (2002) considered planning and control issues in holonic organisations. McFarlane and Bussmann (2000) presented a holonic control mechanism that supports comprehensive manufacturing holons by integrating all control functions into their operations, as illustrated in figure 2.14. In contrast, Sun and Venuvinod (2001) discussed the human side of holonic manufacturing systems, while Akturk and Turkcan (2000) considered cell formation using a holonistic approach. Askin *et al.* (1999) designed and evaluated holonic layouts, where each machine is considered being a holon and randomly distributed throughout the entire facility with no departmental organisations i.e. cells.

### 2.5.3 Fractal manufacturing

A fractal factory, first proposed by Warnecke (1993), applies the theory of fractal geometry to the organisation. The word 'fractal' originates from the Latin word fractus, which means broken or fragmented. In mathematics, the term is used to describe objects

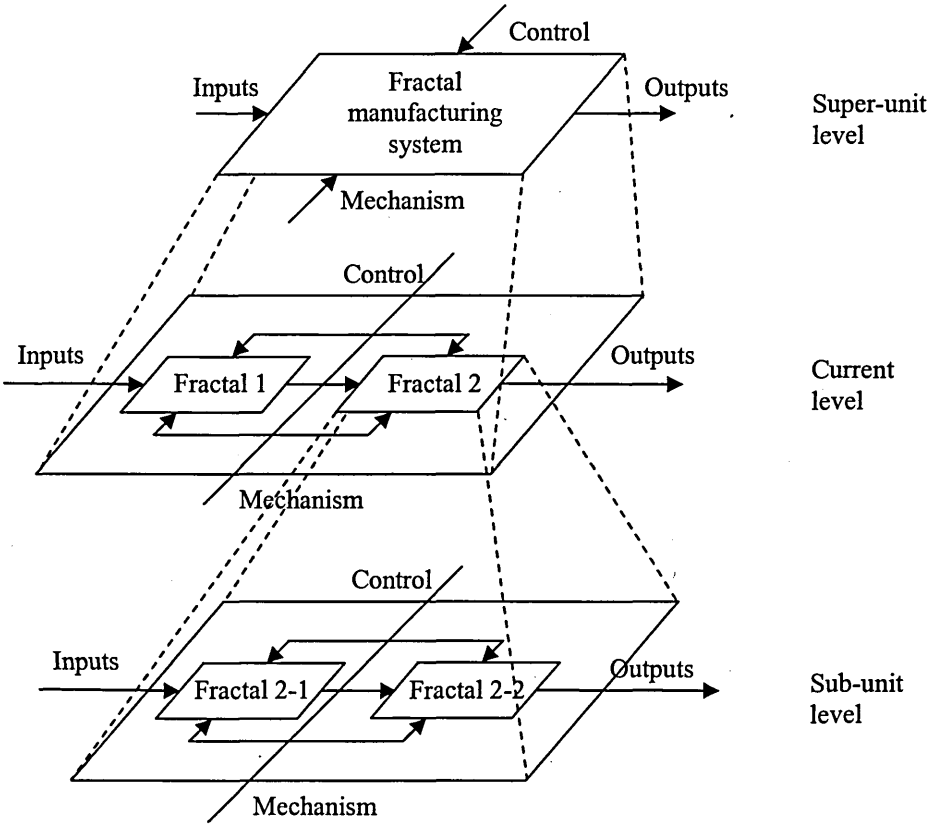



**Figure 2.15** Conceptual structure of fractal manufacturing system (Ryu and Jung 2003)

that replicate their whole structure in each detail when the resolution is increased. Similarly, according to Warnecke (1993), a manufacturing system can be structured using 'fractal units' as basic building blocks of the whole system. In the resulting hierarchical structure a fractal can represent an entire manufacturing system at the highest level or a single machine at the lowest level (Ryu and Jung 2003), as illustrated in figure 2.15. Hence, fractal units are self-similar, self-organising, self-optimising and dynamic 'factories within a factory' (Warnecke 1993). They have the freedom to organise and execute their tasks and select their own methods for problem solving and process improvement. However, at the same time fractals are guided by the same goals, which are generated through coordination between fractals and supported by an inheritance mechanism (Tharumarajah *et al.* 1996). Conflicts are solved through coordination and negotiation, while the dynamic restructuring of processes enables rapid adaptation to changes in the environment (Ryu and Jung 2003). The performance of each fractal is assessed constantly, compared against the target and if a difference is observed necessary adjustments are made (Warnecke 1993). The operation of fractal entities is illustrated in figure 2.16.

Self-similarity is one of the most significant features of fractals as this property defines the structural and operational characteristics of the organisational design (Warnecke 1993). It describes the manner in which fractals perform services, formulate and pursue goals and structure their operations. However since fractals have freedom of execution, the implementation of their goals may differ. This can result in fractals with different internal structures. Nevertheless, fractals are considered to be self-similar if they can

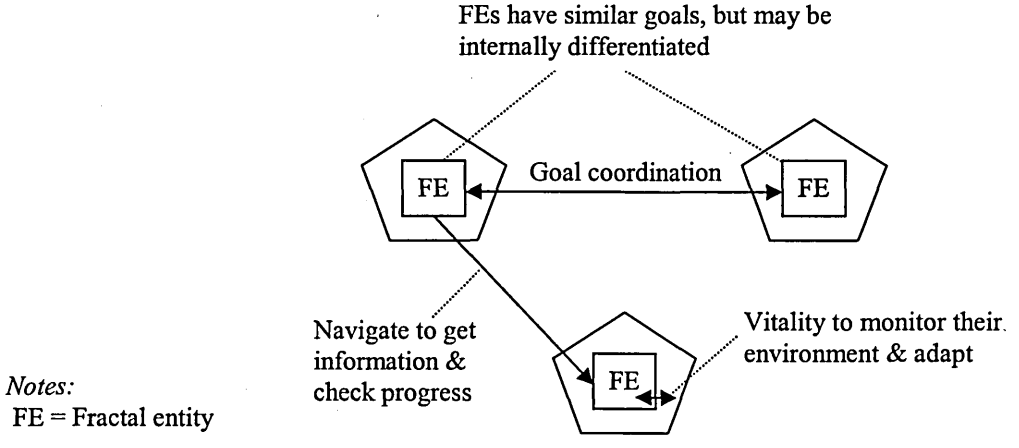



**Figure 2.16** Operation of fractal entities (Tharumarajah *et al.* 1998)

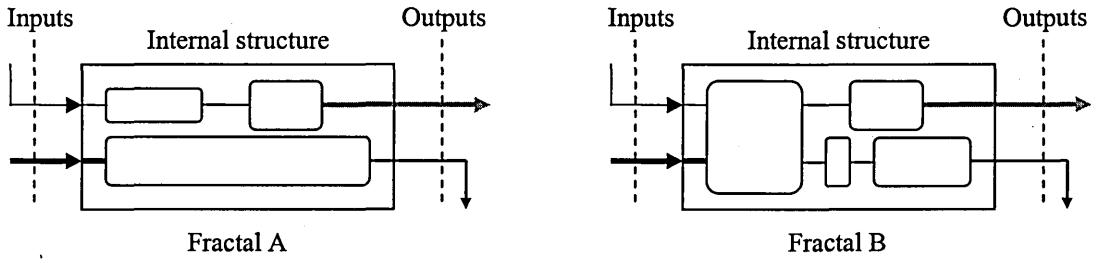



**Figure 2.17** Self-similar fractals with different internal structures (Warnecke 1993)

| Characteristic | Description |
|---|---|
| Self-similarity | Fractals are self-similar and perform services |
| Self-organisation | Fractals practice self-organisation:<br>*operatively*: procedures are optimally organised by applying suitable methods |
| Self-optimisation | *tactically and strategically*: fractals determine and formulate their goals in a dynamic process and decide upon internal and external contacts. Fractals restructure, regenerate and dissolve themselves |
| Goal-orientation | The system of goals that arises from the goals of the individual fractals is free from contradictions and must serve the objective of achieving corporate goals |
| Dynamics | Fractals are networked via an efficient information and communication system. They themselves determine the nature and extent of their access to data<br>The performance of a fractal is subject to constant assessment and evaluation |

**Table 2.4** Basic features of fractal manufacturing systems (Warnecke 1993)

produce the same output with the same input regardless of their internal structures (Ryu and Jung 2003), as illustrated in figure 2.17. The characteristic of self-organisation enables the self-optimisation and dynamic restructuring of fractals. Self-optimisation refers to the control of processes and optimisation of composition, while dynamic restructuring facilitates the reconfiguration of fractals and their networks (Ryu and Jung 2003). Goal-orientation, dynamics and vitality are other important features of fractals. The basic features of the fractal factory are summarised in table 2.4.

To realise the benefits of a fractal factory, Ryu *et al.* (2000, 2001, 2003) and Ryu and Jung (2002, 2003, 2004) proposed a fractal manufacturing system based on the concept of autonomous cooperating agents i.e. basic fractal units. The architecture of a basic fractal unit is illustrated in figure 2.18. It consists of five functional modules, namely

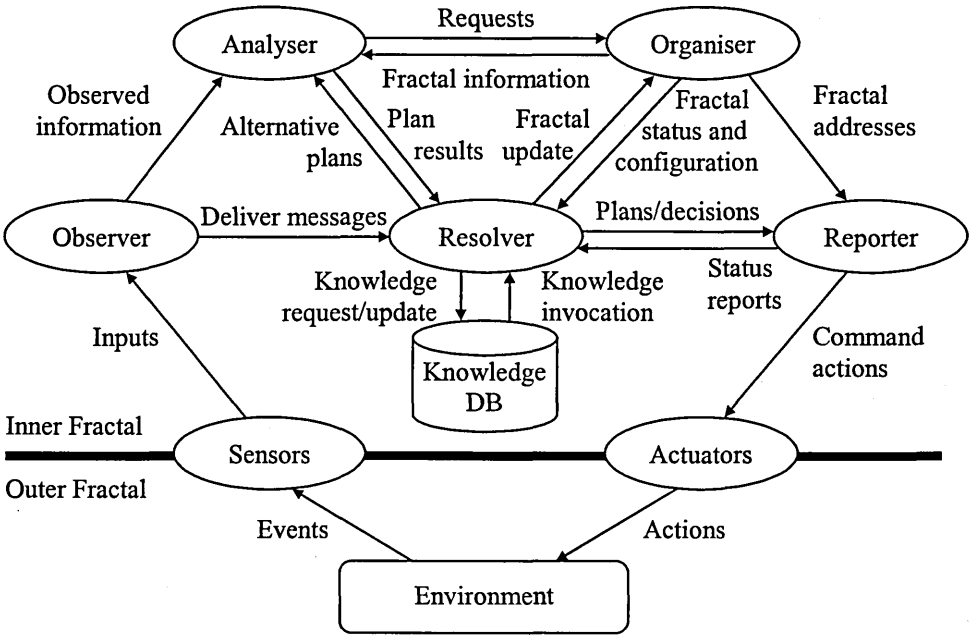



**Figure 2.18** Fractal architecture (Ryu and Jung 2003)

observer, analyser,resolver, organiser and reporter that coordinate and cooperate in order to perform operations in accordance with the fractal goals. Ryu and Jung (2003) designed the basic fractal unit specifically as a general model that corresponds to the elements at any level of the system hierarchy. However, the disturbances in the system and environment can change the system goals. To enable efficient and effective operation under these conditions, Ryu and Jung (2004) developed a methodology for dynamic goal formation in fractal manufacturing systems. An earlier adaptation of a fractal architecture was proposed by Tirpak *et al.* (1992). Other research interests in fractal manufacturing include supply chain management (Noori and Lee 2000, Brehmer and Martinetz 2001, Markfort *et al.* 2000) and layout design (Askin *et al.* 1999, Debnar *et al.* 2001). Venkatadri *et al.* (1997) and Montreuil *et al.* (1999) proposed the concept of the fractal cell, a set of neighbouring workstations on the shop floor, as the basic unit of the organisation for layout design purposes. Sihn and von Briel (1997) examined the process cost and product pricing issues in the control and restructuring of fractal organisations. Kuehnle (1995), Sihn (1995), Strauss and Hummel (1995), Warnecke and Huser (1995) and Klopp *et al.* (1997) focused on promoting the concept of fractal manufacturing, while Tharumarajah *et al.* (1996, 1998) compared the design and operational features of the emerging manufacturing systems, namely fractal, bionic (biological) and holonic, with little contribution to the theory itself. The first application

of the fractal theory was reported by Kimberley (2001) with deployment in a European car company.

### 2.5.4 Biological manufacturing

Biological (or bionic) manufacturing systems were proposed by Okino (1992) and Ueda (1992). The concept aims to apply the structures and behaviour of natural organisms to manufacturing organisations in order to transfer the inherited flexibility and adaptability of life forms to industrial operations. The concept derives from the basic properties of biological systems, i.e. autonomous and spontaneous behaviour, self-development and social harmony within hierarchically ordered relationships (Tharumarajah *et al.* 1996), as illustrated in figure 2.19. The concurrent presence of these properties (in terms of time and location) increases flexibility of system elements towards changes in the environment e.g. diversification of products and abnormal events (Ueda 1992). All biological structures are hierarchically formed with cells as the basic units at the lowest level before ascending to organs, lives and society (Tharumarajah *et al.* 1996). Hence, cells can be seen as the building blocks of biological systems. In addition, all cells have

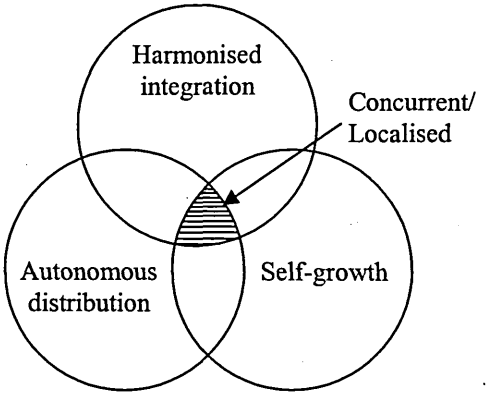
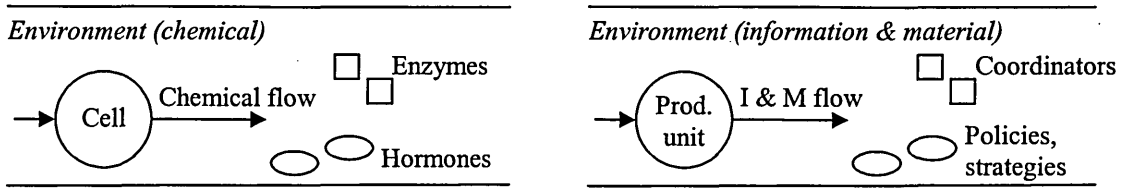



**Figure 2.19** Basic properties of biological manufacturing system (Ueda 1992)




a) cells in biology b) production units in manufacturing

**Figure 2.20** Similarity of biological and manufacturing structures (Tharumarajah *et al.* 1998)

similar structures but different functions and are capable of multiple operations (Tharumarajah *et al.* 1998). Cells exist in a chemical environment which stability is maintained by enzymes and regulated by hormones (Tharumarajah *et al.* 1998). These properties correspond closely to autonomously operating production units on the shop floor, as demonstrated in figure 2.20. The units obtain the inputs from the shop floor, perform operations and return outputs back to the environment. The tasks are specified in a top-down process, while the units' actions at the lower levels support the operation of the whole system in the bottom-up process (Tharumarajah *et al.* 1996).

The ability to adapt to environmental changes and to sustain their own life by functions such as self-organisation, self-recognition, self-growth, self-recovery, learning and evolution are important characteristics of biological systems (Ueda *et al.* 2000). To realise these functions, organisms utilise two types of biological information, i.e. genetic information (DNA-type) and individually learned information (BN-type) (Ueda *et al.* 2000). The biological manufacturing systems employ both types of information to produce products from raw materials, as illustrated in figure 2.21. Vaario and Ueda (1996a) recognised two fundamental approaches of biological manufacturing systems, namely production-oriented and product-oriented. A production-oriented approach views the manufacturing system as a society of individual cells or organisms that collectively respond to environmental stimuli by producing products (Vaario and Ueda 1996a). In a product-oriented view products are seen as organisms that live in the

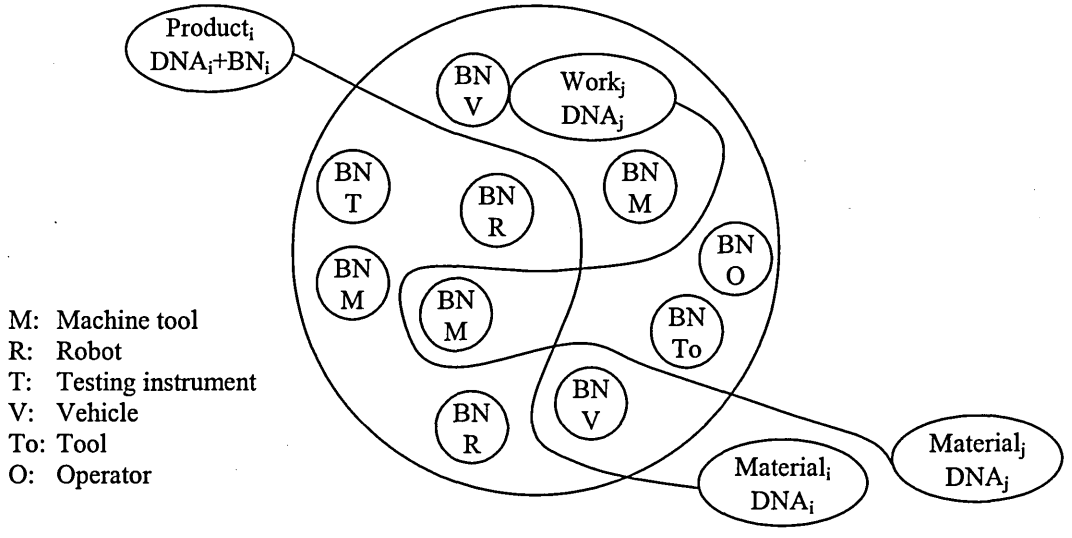



**Figure 2.21** Concept of biological manufacturing system (Ueda *et al.* 2000)

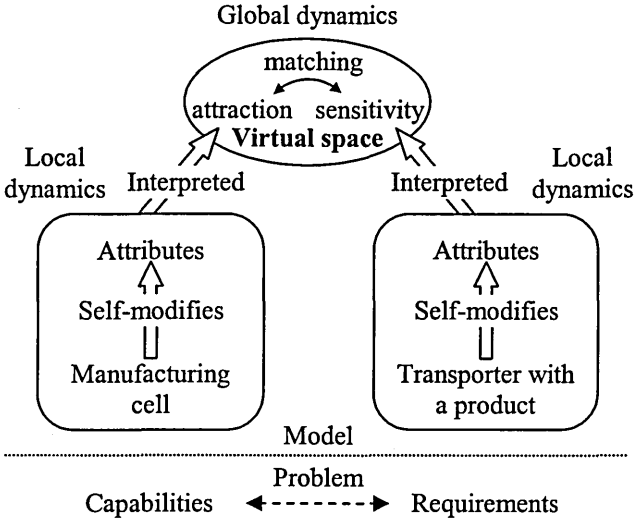



**Figure 2.22** Concept of self-organisation (Vaario and Ueda 1998a)

environment and compete against each other by means of their characteristics determined by genetic information (Vaario and Ueda 1996a).

The most recent research on biological manufacturing systems has focused on realising the self-organising capabilities by proposing dynamic shop-floor configuration (Vaario and Ueda 1996b, 1998b, Fujii *et al.* 1997, Ueda *et al.* 2001b, 2002), reconfiguration (Ueda *et al.* 1997b) and scheduling (Vaario 1996, Ueda *et al.* 1997a, Vaario and Ueda 1998a) methods controlled by a 'self-organisation simulator' demonstrated in figure 2.22. This directs factory operations in real-time by continuously calculating the local potential fields of the machines and transporters on the shop floor (Vaario and Ueda 1998a). According to (Vaario and Ueda 1997) this bottom-up approach leads to a local optimisation with unpredictable global results and enables dynamic and continuous adaptation to disturbances. The problem of scheduling in biological manufacturing systems has been considered by Vaario (1996) and Vaario and Ueda (1998a). They applied the concept of local attraction fields to match and direct jobs to capable and available machines in real-time, creating a dynamic but not necessarily optimal schedule. Vaario and Ueda (1998a) argued that the dynamic scheduling method, which provides real-time control for the shop-floor operations, is competitive in a turbulent environment, because the optimal schedule using global information can be difficult to calculate and maintain. Simsek and Albayrak (2003) summarised the characteristics of a living factory as highly scalable, reconfigurable, flexible, autonomous, cooperative,

intelligent, optimising and reliable. Other research in the area includes modelling of biological adaptation (Vaario 1994a, 1994b) and a reinforcement based learning approach that optimises both, local and global objectives (Ohkura *et al.* 1999, Ueda *et al.* 2000, Fujii *et al.* 2004). Ueda *et al.* (1998) considered the human aspects in biological manufacturing systems, while Brezocnik and Balic (2001), Sluga and Butala (2001), Katalinic and Kordic (2002) and Katalinic *et al.* (2002) proposed alternative models for biological self-organisation. Mill and Sherlock (2000) and Demeester *et al.* (2004) discussed biological analogies in manufacturing. McCormack (2000) reported on an application of a biological manufacturing system that had significantly reduced the operational costs.

### 2.5.5 Responsive manufacturing

The concept of responsive manufacturing refers to the methodology proposed by Gindy *et al.* (1996) for representing product processing requirements and machine capabilities with generic capability units called resource elements to support cell formation. Traditionally, manufacturing cells were formed according to component similarity in terms of the sequence of machines required for the manufacture of the parts (Gindy *et al.* 1996). However, recent improvements to the capabilities of modern machines allow some machines to perform many different operations (Baykasoglu and Gindy 2000). Since the traditional methods of cell formation do not consider these capabilities, Gindy *et al.* (1996) proposed the concept of resource elements (RE) to represent the unique and shared capabilities of machines. The resource element can be described as a collection of form-generating schemata (FGS) that express the basic capability patterns of machining operations regardless of the machine tools used for their execution (Gindy *et al.* 1996). These are formulated by i) identifying the general form-generating capabilities of machining processes (form generating schema), ii) allocating the form-generating schemata to machine tools, and iii) relating the form-generating schemata and machine tools to the specified processing system to form unique resource elements (Gindy *et al.* 1996), as illustrated in figure 2.23. Baykasoglu (2003) summarised the main features of resource elements as i) resource elements being mutually exclusive, i.e. there is no overlap between resource elements, ii) resource elements uniqueness, i.e. sets of FGS in each resource element are different and each FGS can belong to only one resource element, iii) capability to perform all tasks within a resource element available on a resource, iv) ability of a resource to provide one or more resource elements, v) a

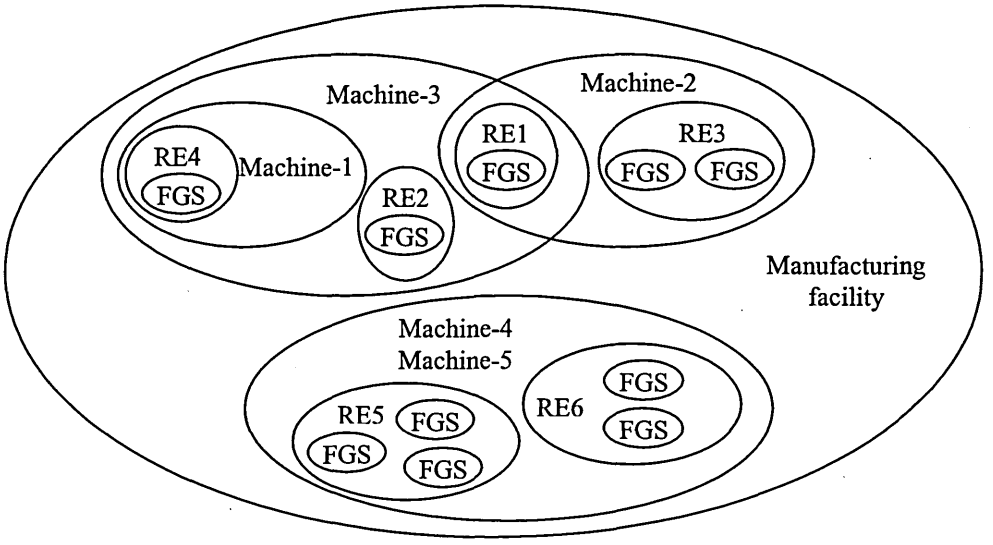



**Figure 2.23** Representation of manufacturing facility using resource elements

component requiring a resource element may access all resources that provide that resource element, and vi) resource elements being unique planning and control entities. Once the machine capabilities and product processing requirements are mapped as resource elements, component groups and cells can be formulated. Gindy *et al.* (1996) claimed that the use of resource elements resulted in a better matching of cells and component groups.

Resource elements were originally designed for component grouping in cellular manufacturing (Gindy *et al.* 1996, Gindy and Ratchev 1997, Baykasoglu and Gindy 2000). However, the recent research has extended the application of the concept to scheduling (Gindy and Saad 1998, Saad and Gindy 1998, Gindy *et al.* 1999, Saad *et al.* 2002), layout design (Baykasoglu 2003) and reconfiguration of virtual cells (Saad *et al.* 2002, Saad 2003). Saad and Gindy (1998) studied the responsiveness of a resource element-based machining facility to disturbances in the environment. They concluded that significant improvements in the system performance can be achieved by representing the machine shop as a set of resource elements. In addition, resource elements enable a manufacturing system to cope with different types of disturbances.

### 2.5.6 Virtual cellular manufacturing

Virtual cellular manufacturing was proposed by McLean *et al.* (1982). This is based on the theory that hierarchical structures that are capable for dynamic reconfiguration of

their sub-systems are able to adapt to changes in the environment (Baykasoglu 2003). The concept combines the benefits of process layouts and traditional cellular manufacturing by creating temporary logical manufacturing cells over functional layouts (Sarker and Li 2001). These virtual cells are formulated in a computer system by allocating machines to logical cells without altering their physical proximity (Sarker and Li 2001). Hence, a virtual cell is not identifiable on the shop floor as a fixed group of machines as it exists only in the system control software as a flexible routing mechanism (Kannan and Ghosh 1996). Virtual cells enable machine sharing between cells that produce different part families with overlapping resource requirements resulting in higher machine utilisations (Irani *et al.* 1993). In addition, the logical nature of the cells enables rapid and flexible reconfiguration of the manufacturing facilities and makes it suitable for a dynamic environment (Baykasoglu 2003). Yusuf *et al.* (2003) noted that virtual cells evolve and dissolve naturally in response to changes in demand mix and volume. They can also reduce production lead times and enhance product customisation (Yusuf *et al.* 2003).

The recent research in virtual cellular manufacturing has focused on cell formation, scheduling and layout design, e.g. Irani *et al.* (1993), Sarker and Li (2001), Baykasoglu (2003) and Ko and Egbelu (2003). Sarker and Li (2001) noted that scheduling in virtual cells is a complex task due to the potential of bottleneck machines overlapping across cells. Mertins *et al.* (2000) made similar observations concerning the capacity assignment of shared resources, while Kannan and Ghosh (1996) studied the influence of shop configuration, set-up time and part mix variability for virtual cell performance. Yusuf *et al.* (2003) analysed the capabilities of virtual cells to handle volume variety. Finally, Saad *et al.* (2002) and Saad (2003) investigated the use of virtual cells for reconfiguration of cellular manufacturing systems. They concluded that the application of the virtual cell concept can improve the performance of cellular manufacturing systems.

### 2.5.7 Next generation manufacturing systems

The next generation manufacturing systems (NGMS) refer to the intelligent manufacturing system (IMS) research programme to develop a global concept on the future of advanced manufacturing systems. The project investigated agile, fractal, biological and autonomous & distributed manufacturing concepts and aimed to integrate

them into a unified framework that supports all aspects of the product related business processes of the future manufacturing enterprise. The NGMS-IM consortium (2000) identified the capability to rapidly manage change through flexible, adaptable, variable, globally-oriented and distributed systems as the most important requirement for the next generation manufacturing systems. They summarised that future organisations would have the key characteristics to i) provide support for virtual enterprises, ii) be reconfigurable, flexible and adaptable in response to customers, iii) be focused on delivering value, iv) be information and knowledge based, but also human intelligence oriented, and v) be modular to support distribution and autonomy, and cooperate to achieve enterprise goals. To meet these requirements NGMS-IMS consortium (2000) proposed the virtual enterprise architecture that is divisible into three layers, namely floor level, factory level, and enterprise level. At the enterprise level, agile manufacturing is employed to provide the vision, while the concepts of fractal factory and biological manufacturing are adopted as implementation methodologies at the factory level. Finally, autonomous & distributed manufacturing systems define the operational building blocks at the floor level, supported by the knowledge and information infrastructure and human resources that form the foundations of the architecture (Choi and Kim 2000). The implementation of the architecture requires i) workforce flexibility, ii) knowledge-based supply chain, iii) rapid product/process realisation, iv) innovation management, v) change management, vi) next generation manufacturing processes and equipment, vii) pervasive modelling and simulation, viii) adaptive, responsive information systems, ix) extended enterprise collaboration, and x) enterprise integration (Choi and Kim 2000).

### 2.5.8 Other emerging manufacturing concepts

Several other less well-known concepts have also been proposed in the academic literature for 21st century manufacturing. They include concepts such as random, multi-channel and reconfigurable manufacturing. Random manufacturing was proposed by Iwata *et al.* (1994) to realise flexible and adaptive production for dynamically changing orders. Its basic characteristics can be summarised as i) autonomous machine system, ii) dynamic machine grouping, iii) tender-based task allocation, and iv) shop floor control through a reward and penalty system. In the random manufacturing system individual machines placed in the same group cooperate to complete tasks while machine groups compete for survival in the environment (Iwata *et al.* 1994). Multi-channel

manufacturing systems were proposed by Meller (1997). These can be understood as a linear formation of fractal cells (Ozcelik and Islier 2003). The approach formulates parallel flow lines with similar and dissimilar machine sequences in order to create alternative routes for significant parts. During production, products can then choose the line that leads to the greatest system efficiency at that time (Meller 1997). Hence, the systems compromise compatibility and specialisation to produce a specific range of products (Ozcelik and Islier 2003). Finally, reconfigurable manufacturing systems were suggested by Koren *et al.* (1999). These are designed for rapid adjustment of production capacity and functionality in response to new circumstances by rearrangement or change of their components (Mehrabi *et al.* 2000). This is achieved through modular structures, rapid integration, quick changeovers, matching system capability to demand, and rapid identification of problems (Mehrabi *et al.* 2000). Reconfigurable manufacturing systems offer a middle ground alternative between volume-orientated dedicated transfer lines and variety-focused flexible manufacturing systems (Mehrabi *et al.* 2000).

## 2.6 Comparison of the emerging manufacturing concepts

All recently proposed manufacturing paradigms aim to solve the same problem, i.e. to enable manufacturing systems to efficiently survive and adapt to a rapidly changing environment (Wyns 1999). However, due to the different origins of the concepts, the approaches and methods they employ to achieve this goal are different (Sousa *et al.* 1999). In addition, some of the concepts have attracted more research than others and include methods and procedures that have been widely accredited in the academic literature. Furthermore, the focus of the research has varied; for example agile manufacturing is mainly a managerial concept that provides vision and strategy for future organisations, while responsive and virtual cellular manufacturing are applied at the shop floor level. Holonic, fractal and biological manufacturing paradigms are more general concepts that can be implemented at all levels of the organisations. Therefore, they can also be comprehensively compared to each other. Several authors have previously compared the design and operational features of holonic, fractal and biological concepts, e.g. Tharumarajah *et al.* (1996, 1998), Sousa *et al.* (1999) and Ryu and Jung (2003). They noted that the underlying principles of these concepts are very similar. Tharumarajah *et al.* (1996) concluded that they all advocate an organisation of

distributed autonomous modules capable of self-organisation to carry out required functions.

### 2.6.1 Organisational features

The holonic, fractal and biological manufacturing systems support both, hierarchical structures and distributed autonomous entities. The hierarchical structure maintains the overall system coherence and objectivity and helps to resolve any possible conflicts between the entities (Sousa *et al.* 1999), while autonomous entities are able to react to unexpected events without assistance from higher levels (Wyns 1999). Hence, hierarchical control in fractal, holonic and biological systems is limited to setting goals and general guidelines, while autonomous entities have the freedom to select their own methods for achieving these goals. In addition, the hierarchical structure itself is not fixed but can evolve with respect to its partners and the environment (Ryu and Jung 2003). Furthermore, the level of autonomy held by the basic units varies between the concepts. Tharumarajah *et al.* (1998) noted that holons and biological cells can self-manage, but their capabilities for self-design or self-governance are limited. Fractal entities on the other hand can be seen as self-governing units that set their own goals and adapt to the environment through dynamic reconfiguration. However, these capabilities can be translated in many different ways for manufacturing applications. According to Tharumarajah *et al.* (1998), fractal systems often apply technology to achieve flexibility, while holonic and biological concepts develop technology to make the devices display autonomous behaviour.

The design of fractal, holonic and biological systems aim for highly flexible and dynamic structures with recurring part-whole relations (Tharumarajah *et al.* 1998). Each basic unit and its functions are predefined at the beginning stage (Ryu and Jung 2003). The function of a holon is formed by dividing the functional requirements of the system into basic holons that maintain their roles throughout system lifetime (Tharumarajah *et al.* 1996). Cell functions in biological systems are formed similarly to holons by using DNA information, but they can be divided into smaller or merged into larger functions during their lifetime as long as the main purpose of the cell remains the same (Ryu and Jung 2003). The creation of a fractal function is more complicated, because in addition to the role of the entity it also includes the immediate environment that it interacts with (Tharumarajah *et al.* 1996). Hence, the functions of fractals can be dynamically changed

i.e. reconfigured or restructured at any time according to the goals of the environment. In fractal manufacturing systems a reconfiguration requires a change to the fractal structure, while a system based on holons re-allocates resources and biological systems

| System parameter | Holonic Manufacturing | Fractal manufacturing | Biological manufacturing |
|---|---|---|---|
| Basic unit | Holon: autonomous and cooperative entity | Fractal: autonomous and self-similar entity, includes the environment | Cell: multifunctional and self-organising entity |
| Autonomy of unit | Highly autonomous: negotiate and cooperate to set goals and tasks, limited by rules | Highly autonomous: define individual goals, adaptable | Highly autonomous: define operations in response to changes in environment |
| Flexibility of unit | React to changes in other holons through cooperation and negotiation | React to changes in environment through dynamic restructuring, self-optimisation and self-organisation | React to changes in environment through self-organisation |
| Creation of unit | Predefined and dynamic: limited to rules and functional decomposition at design time | Predefined: dynamically reproduced or reorganised by self-organisation | Predefined: dynamically reproduced by evolution and self-organisation |
| Unit function | Predefined: holons with required functions can be defined at design time | Initially predefined: can be dynamically reassigned during operation | Predefined: cells with required functions can be defined at design time or divided or merged during operation |
| Definition of group | Holarchy: predefined set of holons to support specific functions | Fractal: predefined set of similar fractals, can be dynamically redefined | Organ: dynamically defined through cell division to support required functionality |
| Autonomy of group | Flexible strategies, fixed rules and stable forms | Inheritance and autonomy of goals, dynamic restructuring | Predefined functions and operational autonomy |
| System re-configuration | Change of resources by reallocation | Change of fractal structure by forming new fractals or reassigning functions | Change of process flows |
| Shop floor layout | Holons are spread throughout the facility, no departmental organisation | Fractal cells: a combination of fractals proportional to whole system | Dynamic self-organisation of cells |

**Table 2.5** Organisational features of holonic, fractal and biological concepts (modified from Tharumarajah *et al.* 1996 and Ryu and Jung 2003)

apply changes to the process flows. At the shop floor level, holonic layouts are formulated by distributing the machines randomly throughout the facility without departmental boundaries (Askin *et al* 1999). In fractal layouts machine types are allocated to a fractal cell in proportion to their quantity in the whole facility (Askin *et al* 1999), while biological layouts are dynamically self-organising (Vaario and Ueda 1996b). Table 2.5 summarises the organisational features of fractal, holonic and biological manufacturing concepts.

### 2.6.2 Operational features

The system elements in holonic, fractal and biological organisations interact with each other both lateral and across hierarchical levels (Tharumarajah *et al.* 1998). Since the systems operate without centralised control these interactions need to be regulated to ensure harmony between autonomous units. All concepts support functional unity through coordination, but they employ different approaches to achieve this. Biological cells use a top-down approach for task specification, while decisions concerning the execution of the tasks follow a bottom-up pattern (Ryu and Jung 2003). For lateral communication and coordination, the cells use their shared environment and coordinators (Tharumarajah *et al.* 1998). The actions are defined by the inputs and outputs of other cells in their environment. Fractals, on the other hand, continuously redefine their goals through the inheritance of global goals and coordination with other fractals using a goal formation process (Tharumarajah *et al.* 1996). They communicate and cooperate with other fractals directly to resolve conflicts and monitor the performance through navigation. Holons formulate broad-spectrum plans (process plans and schedules) at higher levels to ensure the hierarchical coordination of the actions, while the holons at the lower levels make detailed decisions about the execution of the plans by coordination with other holons and provide feedback to higher levels (Ryu and Jung 2003). Table 2.6 summarises the basic operational features of holonic, fractal and biological manufacturing systems.

### 2.6.3 Integration of the concepts

Until now, the majority of research on emerging manufacturing paradigms has considered each concept separately. The first and most extensive project to integrate several concepts has been the intelligent manufacturing system research programme of the next generation manufacturing systems (IMS-NGMS). The project aimed to employ

| System parameter | Holonic Manufacturing | Fractal manufacturing | Biological manufacturing |
|---|---|---|---|
| Autonomy | Handle independently disturbances, cooperate for mutual benefits, governed by fixed rules | Individual goals, cooperate for mutual benefits | Handle independently disturbances, cooperate for mutual benefits, interact with environment |
| Hierarchical coordination | Top-down as partial task specification and bottom-up as decisions and feedback | Top-down and bottom-up as iterative and concurrent goal coordination | Top-down as task specifications and bottom-up as decisions |
| Lateral cooperation | Communication and cooperation among holarchies | Communication, cooperation and navigation among fractals | Indirect communication through a common space, coordination among cells |
| Planning, scheduling and control | General planning at higher level, dynamic and concurrent scheduling and control | Continuous and dynamic through negotiation | Hierarchical planning, dynamic and adaptive scheduling and control |

**Table 2.6** Operational features of holonic, fractal and biological concepts (modified from Tharumarajah *et al.* 1996 and Ryu and Jung 2003)

different concepts at different levels of the organisation. However, there was no integration of multiple concepts at the shop floor level. Few research papers have considered using more than one concept to optimise factory floor operations. Saad *et al.* (2002b) and Saad (2003) applied responsive manufacturing to virtual cellular systems to develop a framework for rapid reconfiguration. They concluded that the system performance can be improved by including virtual cells to the reconfiguration procedure. However, Sousa *et al.* (1999) noted that merging several paradigms into one system should be possible since conceptually different paradigms can cooperate and compliment each other.

## 2.7 Research focus

The conducted review and the comparison of the emerging manufacturing paradigms have provided a detailed account of the state of the research in each concept, its strength and weaknesses and any common and distinctive features. Based on this information, fractal, biological and responsive manufacturing have been identified and selected as the key concepts of the new 21st century manufacturing system. The research focus includes the basic shop floor operations i.e. layout design, scheduling and control. The

notion of fractals is used for system configuration since fractal cells provide an easily definable unit for shop floor fragmentation. Fractal units simplify product routings and reduce lead times, while maintaining flexibility through machine sharing. In addition, smaller units decrease system variability and uncertainty (Ozcelik and Islier 2003). Fractal cells are also multifunctional, flexible and scalable i.e. reconfigurable. Biological self-organisation is used for dynamic shop floor scheduling and control. It enables any decisions concerning the product dispatching, allocation and routing to be made in real-time. Hence, the system can react instantaneously to any internal or external disturbances (Fujii *et al.* 1997). Finally, in order to utilise the full capabilities of the system and improve its flexibility to changes, responsive manufacturing is used for representing both machine capabilities and product processing requirements. Responsive manufacturing is included in the layout design process and shop floor scheduling. Gindy and Saad (1998) supported this approach by stating that the system's performance and its ability to cope with disturbances can be significantly improved if manufacturing facilities are represented and scheduled using the resource element concept.

Together these three concepts create a system, which potentially exhibits the efficiency of fractal layouts, the responsiveness of biological scheduling and the flexibility of resource elements. These capabilities were also emphasised by Ryu and Jung (2003), who noted that the next generation manufacturing system should be an intelligent, autonomous, and distributed system with independent function modules, and have a structure that is flexible, highly configurable and easily adaptable to a changing environment. The proposed framework for the integration of fractal, biological and

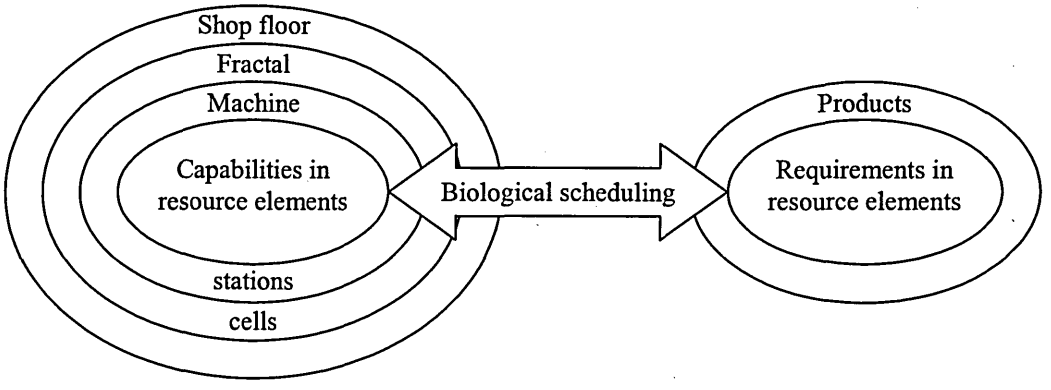



**Figure 2.24** Integration framework for fractal, biological and responsive concepts

responsive concepts for the 21st century manufacturing system is illustrated in figure 2.24. To enable the integration, each concept needs to be further developed regarding the specific functions assigned for them, i.e. fractal layouts, biological scheduling and control, and resource element representation of machine capabilities and product requirements.

# Chapter Three - Research methodology

This chapter describes the methodology employed to answer the research questions. First, the general approach for solving the research problem is determined and justified. Next, the major steps in the research process are summarised. Later in the chapter, the most important tools and techniques deployed during the study, including mathematical and conceptual modelling, heuristic algorithms and computer simulation are reviewed and evaluated. Finally, a hypothetical case study for the implementation, testing and analyses of the developed reference architecture is introduced.

## 3.1 General approach

This research investigates the novel manufacturing paradigms recently proposed in the academic literature. So far, the literature review presented in the previous chapter has revealed that very little in-depth research into these new paradigms has previously been conducted. Hence, the concepts are still relatively abstract in nature with no clear procedure for industrial implementation. This study aims to further develop both, the conceptual framework and the practical applicability of the selected emerging concepts within manufacturing shop floor operations. New procedures and hypotheses are derived from the existing theory and are modelled, simulated and tested using computer simulation techniques. Thus, this study is mainly about testing and applying theories by generating and evaluating quantitative data. This research approach is called deduction. However, the investigation also includes an analysis of the developed system and produced data in order to modify the proposed hypotheses and to generate new theories. Therefore, it also includes inductive characteristics. According to Saunders *et al.* (2003), the adoption of both, deductive and inductive approaches is both common and advantageous in most research projects.

The research is neither a comparative study, nor is it principally about testing existing theories. Instead, a holistic view is taken to find an efficient, responsive and flexible overall structure that employs each of the selected theories or most of their characteristics in different aspects of shop floor operations where they are most suitable. The integrated framework can be understood as a layered model, see figure 2.24, that takes advantage of each manufacturing concept seemingly individually in a single discipline of shop floor operations, whilst other methodologies are built around or on top of it. The general approach aims to reduce the amount of complexity of the experimental research by analysing manufacturing concepts in isolation, each for a different aspect of shop floor operations. However, the research then goes on to integrate the characteristics of each manufacturing concept by building on the results and conclusions of previous experiments. The end result is a combined reference architecture that plays to the strengths of each of the new manufacturing paradigms, and which is supported by objective deductive experimental research. As the generalised theoretical models operate on a high level of abstraction, the experimental research is conducted on a system architecture delivered from a hypothetical case study.

## 3.2 Research process

This research aims to identify and where possible merge the most advanced and distinctive characteristics of fractal, biological and responsive manufacturing systems in order to create an integrated reference architecture for the 21st century manufacturing systems. To meet this objective a specific sequence of activities is performed. Figure 3.1 outlines the major steps of the research process employed in this project. The research starts with a basic review of the academic literature related to the manufacturing systems and operations management to identify research gaps and decide the specific research topic. This is followed by a formulation of the research questions

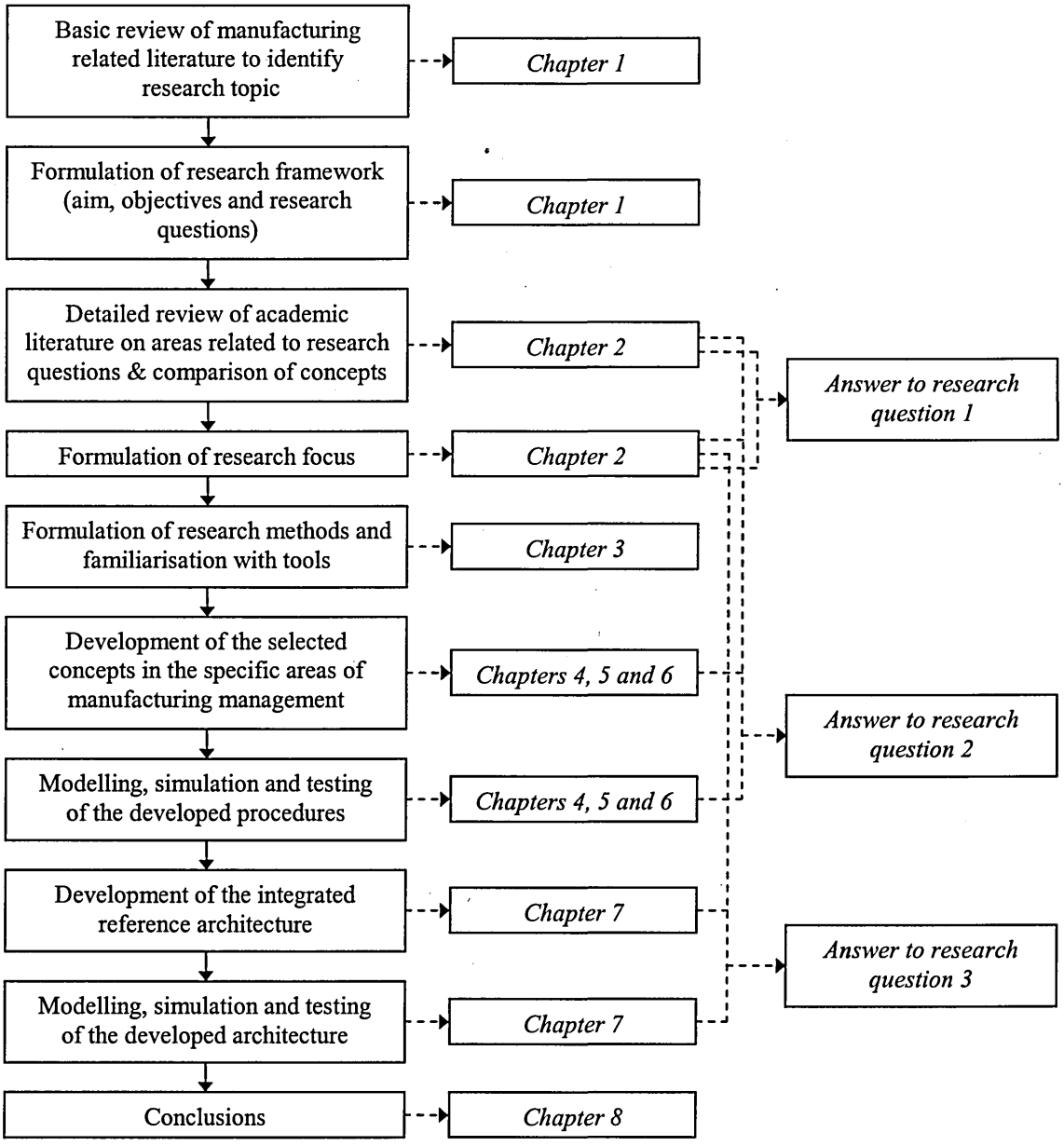



**Figure 3.1** Research process

that form the foundation of the study. Next, a detailed review of the related literature is performed and a comprehensive comparison of the emerging manufacturing paradigms is conducted. Based on the obtained information the research focus is determined and the research methodology decided. After that, the responsive, fractal and biological concepts are studied independently within the context of selected areas of shop floor operations. For each concept procedures are developed, modelled, simulated and tested. Finally, the integration of the concepts is investigated and a reference architecture for the 21st century manufacturing systems is proposed and analysed. The research concludes with proposals for future research.

## 3.3. Deployed tools and techniques

A sample system architecture is selected and modelled with the aim to be representative of real-live scenarios. Experiments are conducted with the aid of computer simulation to evaluate the system when different features of the new manufacturing paradigms are applied to shop floor design and operations.

### 3.3.1 Mathematical and conceptual modelling

Mathematical models are constructed in order to understand some behaviour or phenomenon in the real world, make predictions about that behaviour in the future and analyse the effect various situations have on it (Giordano *et al.* 2003). Ravindran *et al.* (1987) defined a model as a simplified representation of something real. Giordano *et al.* (2003) divided the mathematical models into analytic (symbolic) and replication models. Analytic models use functions and equations to describe a particular phenomenon while replication models attempt to replicate the system behaviour either directly by conducting experimental trials or indirectly by using simulation. Most scheduling problems are NP-hard and it is unlikely to find suitable polynomial algorithms.

Instead, operational research questions regarding layout and scheduling can often be reduced to combinatorial optimisation problems based on replication models. Different combinations or permutations, i.e. in the placement of machines on the shop floor or in the sequence of product dispatch or processing, can be tested and compared to each other in terms of a score to meet an objective function. The evaluation of a specific

layout or schedule is subject to simulation on a modelled system aided by a computer program and results in a numeric value that the system is to be optimised for. The optimisation function is a search conducted on a higher level across a large number of individual simulations with the aim to find the scenario with the lowest or highest result.

### 3.3.2 Heuristic algorithms

An exhaustive search to try all combinations is usually impossible as the complexity of the problem tends to grow exponentially with the amount of elements in the permutation. To find a solution to these problems in a reasonable amount of time a heuristic algorithm can be employed to perform a search to produce good solutions, which are however not necessarily provable to be correct or optimal. The main difficulty of heuristic searches is their inability to escape from infinite loops and to find improved solutions that are significantly different from a locally optimal sequence.

Tabu search is a meta-heuristic method developed by Glover (1989, 1990) and used mainly to solve combinatorial optimisation problems. It is widely used for machine and job scheduling in manufacturing environments (Koylu 2000). The distinctive feature of this higher-level method is its ability to escape a local optimum and guide the lower-level heuristic search algorithm into new areas in order to find the best global solution. This is achieved by allowing moves to neighbouring solutions that are worse than the current solution. Since this can cause a cycling problem to arise, moves to previously visited solutions are forbidden for a certain period of time. These not-allowed moves are recorded in a tabu list. The search process starts from a randomly generated initial solution. Then a neighbourhood of adjacent permutations is generated for the solution using conventional heuristic swap or insert methods. Once solutions are evaluated using the objective function, a move is made to the best admissible solution in the neighbourhood. The search continues until the stopping criterion is satisfied. If several consecutive moves have been performed without improving the best solution, the search can be restarted, either from a known local optimum or from a completely new and random location. Csaszar *et al.* (2000) refer to these two higher-level strategies as intensification or diversification. In contrast to stochastic meta-heuristic methods like simulated annealing and genetic algorithms, tabu search follows a deterministic algorithm, but employs random diversification techniques as a last resort.

### 3.3.3 Computer simulation

The purpose of simulation is to understand the behaviour of a real world system that is captured in a representative but simplified model. Simulation languages can be classed into two major types. Continuous simulation models constantly adjust to dynamic parameters. Discrete-event simulations on the other hand examine a chronological sequence of events that enter a model, and are therefore more suitable for simulations of shop floor operations. These events can be random to introduce the model variability needed to investigate system flexibility, adaptability and control. The introduction of random elements causes non-deterministic results, but most languages allow values to be within fixed limits and proportions.

#### 3.3.3.1 General-purpose object-oriented programming languages

General-purpose programming languages offer the good amount of flexibility, but program development is a slow and highly complicated process and logical errors are hard to avoid completely. It was necessary to use a flexible programming language to optimise a simulated shop floor layout through heuristic search over permutations, because no suitable special-purpose language was available. C++ as general-purpose object-oriented programming language was selected for experiments on shop floor layout optimisation using fractal cells. It offered the needed flexibility to run heuristic searches and the ability to closely model elements such as machines and cells with their associated functions to abstract objects defined in class structures.

#### 3.3.3.2 Special-purpose simulation languages and software packages

Special-purpose programming languages dedicated to simulation provide a high-level framework with which a designer can build specific modelled architectures in a short space of time. Much of the complexity of programming languages remains hidden from the user, but the level of flexibility is severely reduced. The ARENA simulation package as the most widely used sequential simulation tool for this application (Yapa 2003) was selected for most of the experiments on scheduling and control. It is based on the discrete-event simulation language SIMAN and offers a limited programming interface where increased flexibility is needed.

## 3.4 Hypothetical case study

For the implementation, testing and analysis of the developed methodologies and the proposed reference architecture a hypothetical case study was introduced. The case study involved an industrial company that manufactured a high variety of products in low quantities in a highly competitive business sector. Consequently, the company had to cope with frequent and unpredictable small batch size orders from various customers that required short delivery times, customised products and low prices. The demand for the company's products together with their processing times and sequences are summarised in table 3.1. It shows a total demand of 200 products in 10 different varieties for a period of 100 time units. All orders were assumed to arrive simultaneously in a batch size of one unit at the beginning of the time period (at time zero) and were set to have the same priority and due date of 100 time units. In addition, all products were assumed to be approximately of the same physical size, requiring the same effort when handled.

To meet the demand the case study used five different types of machine tools (A, B, C, D and E) that performed a wide variety of machining operations. The availability of each machine was assumed to be 90% of the total period of 100 time units. In addition, it was assumed that the setup of machines between different product types required no time delays or operator intervention. To satisfy the demand and due date requirements, the functionally arranged shop floor required a total minimum of 32 machines of types A, B, C, D and E with individual quantities of 6, 8, 2, 5 and 11, respectively. All

| | | | Processing times in machines | | | | |
|---|---|---|---|---|---|---|---|
| **Product** | **Sequence** | **Demand** | **A** | **B** | **C** | **D** | **E** |
| 1 | A,E,B | 25 | 7 | 3 | | | 10 |
| 2 | A,C,E | 15 | 3 | | 2 | | 11 |
| 3 | E,D,B,A,C | 40 | 2 | 9 | 1 | 2 | 5 |
| 4 | B,D,A | 10 | 5 | 8 | | 6 | |
| 5 | C,D,B,A,E | 15 | 4 | 2 | 3 | 6 | 3 |
| 6 | B,C,E,D | 15 | | 3 | 1 | 5 | 2 |
| 7 | D,E,B | 30 | | 2 | | 1 | 3 |
| 8 | A,B,E,D | 5 | 8 | 2 | | 3 | 6 |
| 9 | E,A,B,C,D | 30 | 2 | 2 | 1 | 3 | 1 |
| 10 | C,E,A | 15 | 1 | | 1 | | 5 |
| Total | | 200 | 525 | 720 | 175 | 440 | 915 |

**Table 3.1** Product demands and processing sequences and times in the case study

replicates of the same machine type were assumed to be identical and therefore any product could be processed at any machine instance with the same efficiency. Each machine had an input and output queue size of one product and occupied the same physical dimensions on the shop floor. Moreover, it was assumed that the system required no operators, resource breakdowns never happened and that the quality of the finished products was perfect (no rework required).

The products were transported in the system using a maximum of 25 automated guided vehicles or AGVs, each capable of carrying one product at a time with a speed of up to 40 network zones per hour. The parts were dispatched to the system in a random order as soon as an AGV became available for pick-up. Loading or unloading of products caused no time delay. The transporters moved to parking areas located at the dispatcher and exit to recharge batteries when not needed.

# Chapter Four - Responsive module development

This chapter reports on the development and application of the responsive manufacturing paradigm in the context of dynamic self-organisation based scheduling systems. First, the relationship between manufacturing flexibility and responsiveness is examined. Then the flexibility capabilities of the resource element based representation of machine capabilities and product processing requirements are investigated. Subsequently, a procedure for the application of resource elements in the self-organisation based scheduling is proposed. The methodology is then modelled and simulated using Arena simulation software. Finally, a number of experiments are conducted using the hypothetical case study.

## 4.1 Manufacturing flexibility and responsiveness

In today's unpredictable and rapidly changing marketplace an organisation's ability to respond to changes in customer demand has become the key for market success (Vokurka and O'Leary-Kelly 2000). Manufacturing flexibility has been widely recognised as the critical component in achieving this. The flexibility can be defined in many different ways, but in the short-term it means the ability to adapt to changing conditions using the existing set and amount of resources (D'Souza and Williams 2000), where changes include both external factors e.g. demand variations, and internal disturbances e.g. machine breakdowns (Kochikar and Narendran 1998). Several authors have attempted to define the dimensions of manufacturing flexibility with varying results, e.g. Koste and Malhotra (1999) identified ten dimensions of flexibility, Vokurka and O'Leary-Kelly (2000) suggested 15 dimensions, whereas Zhang *et al.* (2003) proposed seven elements of manufacturing flexibility. Some of the most commonly quoted flexibility types are machine, routing, mix, volume and product flexibilities. D'Souza and Williams (2000) categorised manufacturing flexibility elements into an externally-driven and an internally-driven dimension, while Yusuf *et al.* (2003) argued that there are only two principle dimensions of flexibility in manufacturing, product mix and product volume, while any other types of flexibility are merely derived from them. On the other hand, Pagell and Krause (2004) argued that there is no evidence that higher levels of flexibility in a volatile environment improve system performance.

Routing, machine and material transfer flexibilities have been identified as being critical to system operation (Kochikar and Narendran 1998). Routing flexibility is the ability to process a given set of products using multiple alternative routes (Zhang *et al.* 2003). It can be improved by increasing the number of identical machines in the system, employing multipurpose machines or extending the material handling system (Tsubone and Horikawa 1999). The availability of alternative routings have been recognised to enhance a system's flexibility towards changes in product mix and disturbances as it enables products to be processed at alternative machines in the case of machine breakdowns (Zhang *et al.* 2003). In addition, Kochikar and Narendran (1998) noted that routing flexibility may improve system throughput, utilisation and load balancing and reduce product transfers. However, there has been disagreement concerning its impact on scheduling. Tsubone and Horikawa (1999) noted that routing flexibility allows for

efficient scheduling, while Koste and Malhotra (1999) argued that a greater variety of routing options complicates scheduling. Another important dimension of flexibility from an operative point of view is machine flexibility. It can be defined as the capability of a machine to perform different operations (Tsubone and Horikawa 1999), and is often a key variable in shop scheduling (Koste and Malhotra 1999). Changeovers between operations normally require a penalty e.g. machine setup.

## 4.2 Resource elements and flexibility

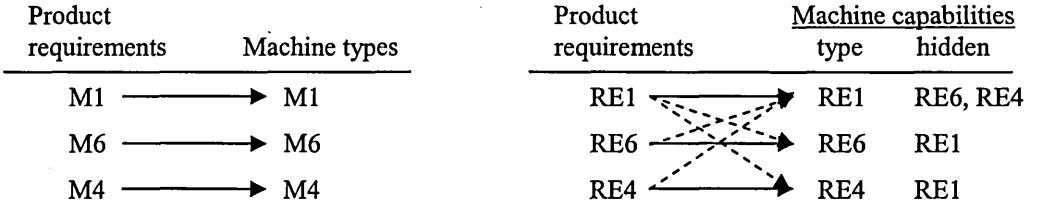
The application of the concept of resource elements to a manufacturing system aims to utilise any hidden capabilities of machines and improves machine and routing flexibility. The recent increase in the scope of capabilities of modern machines has made the resource element-based approach viable to many manufacturing systems (Gindy *et al.* 1999). The traditional approach that described component processing requirements as a sequence of machines required for their manufacture is unable to utilise multiple capabilities of machines. Hence, the resource element-based representation of exclusive and shared capabilities of a system's resources can increase the number of alternative routings available in the system and improve resource utilisation. Gindy *et al.* (1996) noted that the flexibility of a manufacturing system is determined through the number of duplicate resources and the similarity and uniqueness of the resources in the system, while Gindy *et al.* (1999) argued later that system responsiveness can be improved by maximising the utilisation of any flexibility inherent in its available resources.

Scheduling in resource element-based systems has been considered by Gindy and Saad (1998), Saad and Gindy (1998), Gindy *et al.* (1999) and Saad *et al.* (2002). They employed a simulation optimisation model to find a satisfactory schedule for cellular manufacturing systems. The procedure used dynamic shop floor information during the optimisation and a schedule based on the global view of the system resources was developed. Saad and Gindy (1998) concluded that significant improvements in system performance can be achieved if machine capabilities are represented as resource elements. However, the application of resource elements in recently introduced self-organisation based scheduling approaches that employ only local information has not been investigated.

## 4.3 Proposed self-organisation-based scheduling using resource elements

Self-organisation refers to the bottom-up approach to generate individual systems and independent behaviour through interaction of basic components at the lower levels (Vaario and Ueda 1998a). Within scheduling it means that machines, transporters and orders negotiate to determine the product routings based on local conditions on the shop floor at that moment. No advanced scheduling or sequencing of orders takes place. Traditionally this idea was implemented according to machine type information. However the lack of precise, advance information may lead to temporary uneven demand of resources due to the sequence of products not being optimised. The introduction of resource elements would enable a more even distribution of load across a larger number of capable resources, as illustrated in figure 4.1, thus avoiding temporary bottlenecks. It would also increase a system's ability to cope with internal and external disturbances by introducing alternative routings. However, resource elements introduce an additional setup delay to the system when processing requirements at the machine change, whereas setup delays traditionally only occurred when a product of a different type entered the machine. Hence, there might be a need to restrict the use of hidden machine capabilities. Basic rules need to be formulated to reach a decision in this trade-off scenario of bottleneck avoidance against setup delays. The following rules have been established to guide the use of hidden capabilities of machines as represented by resource elements.

1. Always route the product to a machine of required type if available.
2. If no machine with the required resource element is available, then
   2.1 route to the machine with the required resource element that has the shortest queue, or
   2.2 wait until a machine with the required resource element becomes available.




a) machine types b) resource elements

**Figure 4.1** Matching product requirements and machine capabilities

## 4.4 Implementation and results

The proposed approach to resource element-based self-organisation in scheduling was applied to the hypothetical case study introduced in chapter 3. The purpose was to evaluate the performance of the resource element-based approach to disturbances and to compare it to a traditional machine-based scheduling system. Table 4.1 summarises the randomly selected machine capabilities as represented by resource elements. The simulation model was developed using Arena simulation software. The program code for the model is presented in appendix A. The model assumes no setup delays or product transport delays. The following experiments were conducted.

1. Steady state (no changes to product inputs or machines).
2. One replicate of machine type C becomes unavailable throughout the simulation.
3. Minor changes in the input product demand (demand for product type 1 was set to zero and the demand for product type 2 was doubled to 40).

The outcome of the experiments indicated that the resource element-based self-organisation system was able to cope better with changes to machine availability and demand mix than the traditional machine-based system. Under stable conditions both systems performed well and were able to comply with the due date requirements as indicated in figure 4.2. However, when internal or external conditions changed, the resource element-based self-organisation was able to adapt to the new circumstances while the machine-based self-organisation could not. This is due to the fact that the machine-based system was not able to utilise the hidden capabilities of resources. Figures 4.3 and 4.4 illustrate similar performance in terms of the average product lead times and resource utilisation. The resource element-based system has a longer lead

| Machine-based system | Resource element-based system 1 | Resource element-based system 2 |
|---|---|---|
| A (RE1) | RE1, RE2 | RE1, RE2, RE3 |
| B (RE2) | RE2, RE3 | RE2, RE3, RE4 |
| C (RE3) | RE3, RE4 | RE3, RE4, RE5 |
| D (RE4) | RE4, RE5 | RE4, RE5, RE1 |
| E (RE5) | RE5, RE1 | RE5, RE1, RE2 |

**Table 4.1** Machine capabilities in the case study using resource elements

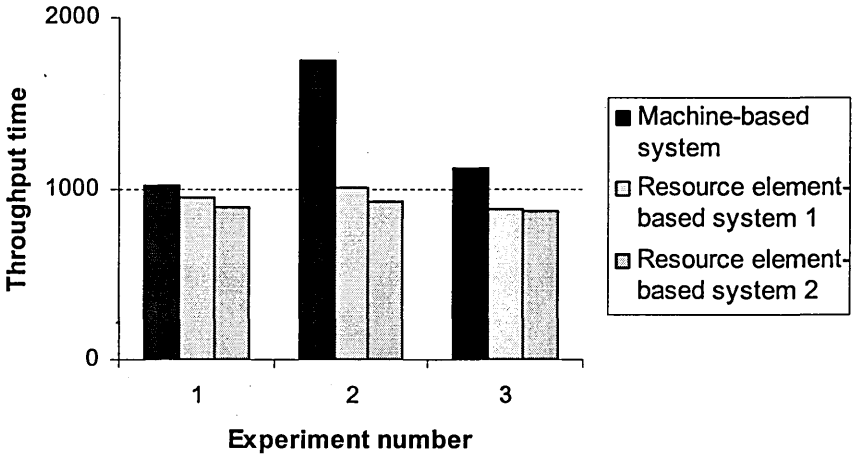



**Figure 4.2** Throughput times in machine and resource element-based systems

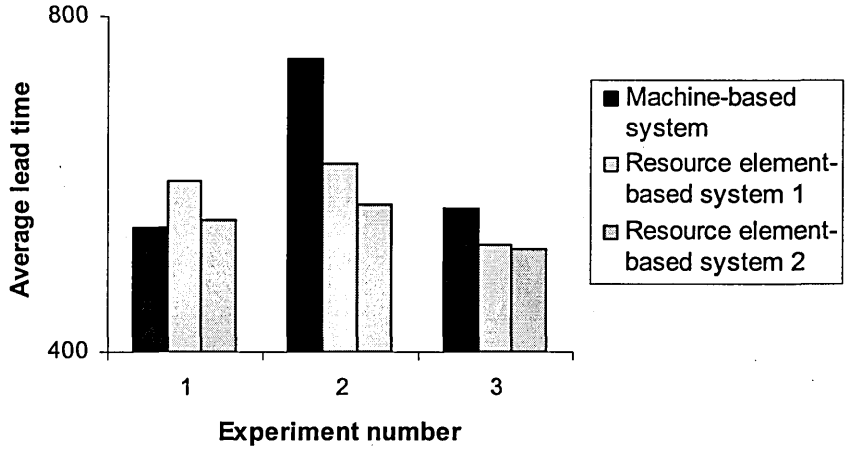



**Figure 4.3** Lead times in machine and resource element-based systems

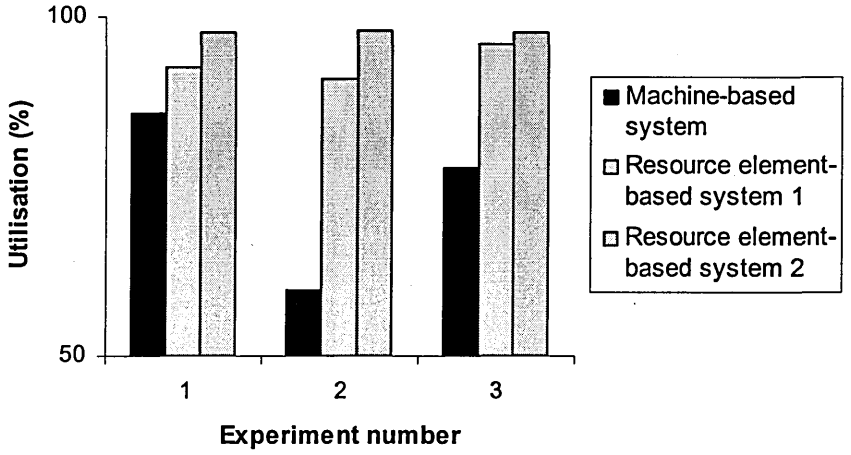



**Figure 4.4** Utilisation in machine and resource element-based systems

time in a steady state system, but once the internal conditions vary from the optimum or demand mix changes from the values originally used for system design, it outperforms machine-based systems. The ability of the resource element-based system to exploit the hidden capabilities of resources improved average utilisation, as illustrated in figure 4.4. These results indicate that resource elements can be applied successfully to self-organisation systems.

## 4.5 Conclusions

The results of the experimental study indicate that resource elements are applicable to dynamic self-organisation scheduling systems. The proposed approach successfully balanced the load among all capable machines when a machine breakdown or changes in the demand mix lead to higher demand on some resources. It also enabled the system to meet the due date requirements. Hence, resource elements can improve system flexibility and responsiveness in the conditions of dynamic self-organisation and are suitable for 21st century manufacturing systems.

# Chapter Five - Fractal module development

This chapter proposes a methodology for shop floor configuration in fractal manufacturing systems. It includes an overview of the layout design process and a discussion on the system re-configurability. It also summarises the existing fractal layout design approaches. Later in the chapter, seven distinct fractal cell configuration methods for different system design objectives and constraints are proposed and an integrated design methodology is developed. In addition, mathematical models for the different design stages are presented. The procedure is modelled and simulated using heuristic algorithms and c++ programming language. Finally, the procedure is applied to the hypothetical case study. The quality of the resulting layouts is assessed and compared against the random arrangement of machines. The chapter ends with a flexibility study.

## 5.1 Layout design

The discipline of layout design in manufacturing is concerned with the determination and allocation of the available space to a given number of resources (Azadivar and Wang 2000). It is a complex task that can significantly affect the efficiency of the resulting manufacturing system in terms of shop floor control, materials handling, materials management, equipment utilisation, and worker productivity (Co and Araar 1988). Traditionally, layout design has focused on minimising the amount and distance of material transportation due to the relatively high costs of material handling (Gau and Meller 1999) and on maximising the capacity utilisation (Wild 1993). It involves consideration of the demand, capacity, work methods, resource requirements, handling and movement, departmental area requirements, and shape and location restrictions (Wild 1993, Gau and Meller 1999). Koren *et al.* (1998) summarised the parameters of manufacturing configurations as i) the system initial cost, ii) quality, iii) reliability and throughput, iv) scalability i.e. the cost of adding capacity to adapt to market demand, v) the number of product types that the system can produce, and vi) the system conversion time between products. In the design of conventional layouts such as product, process and cellular layouts, the product mix, volume and routings are generally assumed to be known and valid for a long period of time (Benjaafar *et al.* 2002). In addition, the employed algorithms focus solely on material handling efficiency and ignore the flexibility and re-configurability of the ensuing layouts.

Recent changes in customer demand patterns have increased the pressure on manufacturers to produce a wider variety of products with shorter cycle times. This requires resource configurations that can either function efficiently over many different production scenarios or that can easily be reconfigured (Benjaafar *et al.* 2002). Meng *et al.* (2004) classified the layout problems into traditional, robust, dynamic and reconfigurable according to the availability of production data for single or multiple production periods. Robust layouts have been designed to operate efficiently in multiple production scenarios, while dynamic layouts balance the costs of material handling and reconfiguration over a sequence of layouts designed for multiple planning periods (Meng *et al.* 2004). The reconfigurable layout approach assumes that resources can be easily moved around the shop floor, thus making frequent relocation of resources feasible (Baykasoglu 2003). Baykasoglu (2003) considered four types of layout

strategies, namely modular, reconfigurable, agile and distributed (scattered) layouts. Modular layouts group machines according to a subset of operations in different routings, while agile layouts are designed according to agility criteria e.g. throughput, cycle time etc. In distributed layouts the machines are scattered throughout the shop floor to improve the accessibility from different areas of the layout (Baykasoglu 2003). Fractal and holonic layouts can be considered as different types of distributed layouts. Askin *et al.* (1999) developed machine distribution and product routing methods for holonic layouts, where each machine is considered as an independent entity and the resources are scattered around the borderless shop floor to provide efficient product routings for any product type. The methodologies proposed by Benjaafar and Sheikhzadeh (2000) and Urban *et al.* (2000) optimise the machine arrangement in a distributed holonic layout for a known product demand. A system with a fractal layout, on the other hand, uses cells to group machines together and to control and limit product routings. Although the fractal layout can be seen as an extension of a cellular layout (Askin *et al.* 1999) due to the structure of the shop floor, fractal cells are multifunctional and able to process most of the product types routed into the system. Multi-channel layout studied by Ozcelik and Islier (2003) can be seen as the linear formation of fractal cells. Benjaafar and Sheikhzadeh (2000) explored the design of flexible layouts i.e. layouts capable of a cost-effective operation over a number of demand scenarios, by creating replicates of the same department and distributing them throughout the shop floor. They discovered that having duplicate departments strategically located around the facility can simultaneously reduce material handling costs and improve responsiveness towards demand fluctuations. The facility layout optimisation problem has been widely reported in the academic literature and many different optimisation procedures have been proposed e.g. Lacksonen (1997), Balakrishnan and Cheng (1998), Azadivar and Wang (2000), Lee *et al.* (2001) and Urban *et al.* (2000).

## 5.2 Layout design in fractal organisations

The first methodology for applying the theory of the fractal factory to facility layout design was proposed by Venkatadri *et al.* (1997) and Montreuil *et al.* (1999). The authors suggested the use of a 'fractal cell', a set of neighbouring workstations on the shop floor, as the basic unit of the organisation. In their approach, all fractal cells had roughly the same composition of machines and were capable of processing most of the

demanded products; hence system flexibility was believed to increase. The proposed method first determined the required capacity levels for each machine type and the number and composition of fractal cells. Then an iterative algorithm was employed, which continuously optimised the layout and flow assignment according to the performance of the system under these parameters. The objective was to create a workstation layout that minimises the capacity requirements and material travelling distances for a known product mix and demand. The computational results indicated that unrestricted product flows offer the best flow scores in a fractal layout. Venkatadri *et al.* (1997) argued that independent cells would be competitive only if product specialisation was allowed. On the other hand, if the products were to be distributed evenly among cells, free routings over cell borders would be required for the minimisation of material handling distances. The authors claimed that in an agile environment similar cells would offer several advantages, such as easy control and expansion, and operational flexibility, but would suffer either from excess capacity or long product travelling distances.

Askin *et al.* (1999) used a slightly different approach to the fractal layout design. In their experiments all fractal cells were identical, fully independent, and capable of processing all products. In addition the machines were located randomly within each fractal, e.g. cells were not specialised for any product. The fractal layouts for a different number of machines and processes were developed and simulated in a chaotic environment with randomly generated processing sequences and times and varying product inter-arrival times. When the authors compared the results to holonic and process layouts, they concluded that the fractal layout, which used the total workload of relevant machines for a fractal selection, had both the smallest cycle time and material handling distance. This supports the main objective of fractal layout, which is to reduce material movements by forming small multifunctional cells with short part routes (Askin *et al.* 1999).

While Askin *et al.* (1999) presented an overall convincing case for independent and similar fractals, they only considered a situation where all cells had exactly the same composition of machines. This rarely exists in a manufacturing system and is likely to require resource duplication. On the other hand, the methodology proposed by

Venkatadri *et al.* (1997) created layouts using known product demand and mix. Unfortunately, in an agile environment the exact product mix and demand levels are very difficult to predict. Therefore, the created layout could quickly become ineffective and might require reconfiguration. In addition, the material flows in the resulting layouts are very complex and difficult to control. Since the authors (Venkatadri *et al.* 1997) maintained the cellular separation of fractal cells only during the initial system design, virtual manufacturing was suggested for the actual system operation and control. Although allowing free or almost free product routings over cell borders can reduce the product travelling distances, many significant features of fractal manufacturing are consequently lost.

Montreuil *et al.* (1999) discussed product distribution and routing for identical and for different fractal cells. The authors recognised that identical cells with the same machine compositions and layouts enable the processing of all products in all cells with the same efficiency. They also noted that for identical cells the material travelling distances can be reduced by optimising the layout in each cell according to the allocated share of the product mix. The third layout option Montreuil *et al.* (1999) considered allowed different cell compositions, which they claimed would generally require inter-cell flows to enable the processing of all product types in all cells.

## 5.3 Proposed fractal layout configuration methods

Fractal cells can take many different forms, as illustrated in the review of fractal cell layouts. This is due to the multitude of interdependent design parameters, such as the level of interaction between cells, the similarity of cells in terms of machine types and quantities, the system capacity level and the distribution of product types among the cells. A comprehensive review of the fundamental design parameters is presented in table 5.1. Since many of the design parameters have a fundamental influence on the structure and operation of the system, they have to be in line with the strategic goals of the organisation. Accordingly, no single type of fractal layout can be recognised as an optimal solution for every organisation and tradeoffs may have to be made. From all possible combinations of the fractal cell parameters only seven distinct cell configuration methods were identified to be useful from a business perspective and selected for experimental analysis.

| Design parameter | Key issue | Options |
|---|---|---|
| Cell autonomy | Can products be routed between cells? | • Autonomous cells<br>• Semi-autonomous cells (cooperation allowed on pre-defined routes only)<br>• Cooperative cells |
| System capacity | Can the number of resources in the system exceed the minimum quantity required by the demand? | • Minimum capacity<br>• Limited resource duplication allowed (for performance improvement only)<br>• No capacity restrictions |
| Similarity of cell compositions | Can all cells process all products? | • Identical cell compositions<br>• Similar cells compositions (machine quantity not divisible by number of cells)<br>• Different cell compositions (optimised for a fraction of the demand) |
| Similarity of cell layouts | Can all cells process all products with the same efficiency? | • Identical cell layouts<br>• Similar cell layouts (slightly different cell compositions)<br>• Different cell layouts (optimised for a fraction of the demand) |
| Demand allocation | Can any product be allocated to any cell? | • Even distribution of demand to cells<br>• Optimised distribution of demand to cells (according to cell compositions and/or layouts) |
| Number and size of cells | Which cell quantity results in an optimal cell size? (small cell quantity = large size cells, and vice versa) | • Suggested range: any cell quantity between one and the largest number of machines of one type $1 \leq F \leq \max(N_m)$<br>• Approx. all machines divided by the number of machine types ($N/M$)<br>• Smallest number of machines of one type $\min(N_m)$ (allows full capabilities in all cells without duplication) |
| Shape of cells and shop floor | Which cell and shop floor shape offers minimal material handling distances? | • Square-shape<br>• Rectangle-shape<br>• L-shape<br>• Any other shape |

**Table 5.1** Fundamental design parameters of fractal cell configuration

The first fundamental design classification as illustrated in figure 5.1 follows the decision of how to distribute products to the cells. In methods 1 to 3 the products are assigned to the cells as evenly as possible to facilitate easy control and responsiveness to disturbances. Conversely, in methods 4 to 7 the products are assigned to cells according to cell capabilities in order to minimise capacity increase or the number of

external material flows. In method 7 only the capabilities of non-specialised cells are considered during the product assignment.

The allocation of machines to fractal cells is a more complex issue relating to capacity planning, cell similarity and cell autonomy. In terms of cell compositions, the proposed methods 1, 2 and 3 are directly comparable to methods 4, 5 and 6 (identical, similar/optimised, minimal). Resource allocation is simplest in methods 3 and 6, for which merely the minimum required machine quantities, as defined by assuming a one cell system, are distributed as evenly as possible to all cells. Configuration methods 1 and 2 base the allocation of machines strictly on each cell's resource requirements as defined by the products assigned to it. Since the product distribution in these methods is similar, the cell compositions tend to be similar as well. Methods 4 and 5 start with minimal resource allocations analogous to methods 3 and 6. However, if no means of product allocation can be found in which all cells are able to independently process the assigned demand, then machines are added as required. Resources are additionally duplicated in methods 1 and 4 to achieve identical cell compositions. In method 7 a specialised cell is first created to contain specialised or scarce resources which quantities are smaller than the number of fractal cells. Then the remaining resources are assigned evenly among the normal cells. As in methods 5 the number of these resources is then increased until all products can be distributed without requiring cross-cell flows

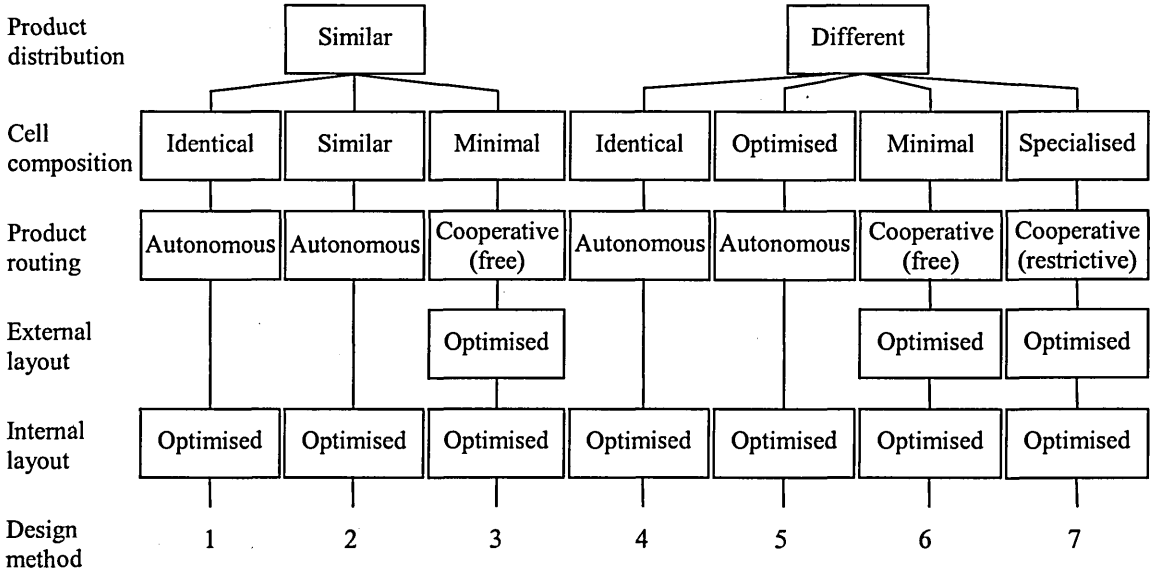



**Figure 5.1** Fractal cell configuration methods

to occur between normal cells. Machine and product allocations in the proposed fractal cell configuration methods are illustrated in figure 5.2.

The composition of fractal cells is strongly affected by capacity planning, which must balance investment cost against operational benefits. In general, excess capacity should be avoided and machines should only be added if a significant improvement on the performance or processing capability of the system can be gained. Forcing identical cell compositions by resource duplication, as required in methods 1 and 4, may result in a high degree of overcapacity on some resources, while the utilisation of others might remain high. If different cell compositions are allowed, they often occur as a consequence of varying processing requirements and machine quantities, even though they are generally not aimed for. In any case, the available resources should be distributed as evenly as possible among the cells to increase system flexibility, robustness, adaptability and capability. In some instances the business structure may favour layouts with different cell compositions, i.e. if group technology can be applied or if machine requirements make the formation of efficient identical cells difficult or infeasible. When cells are similar, product re-distribution in the case of cell malfunction is easier. If cells are independent, it is important to consider the cell capabilities before products are routed to cells.

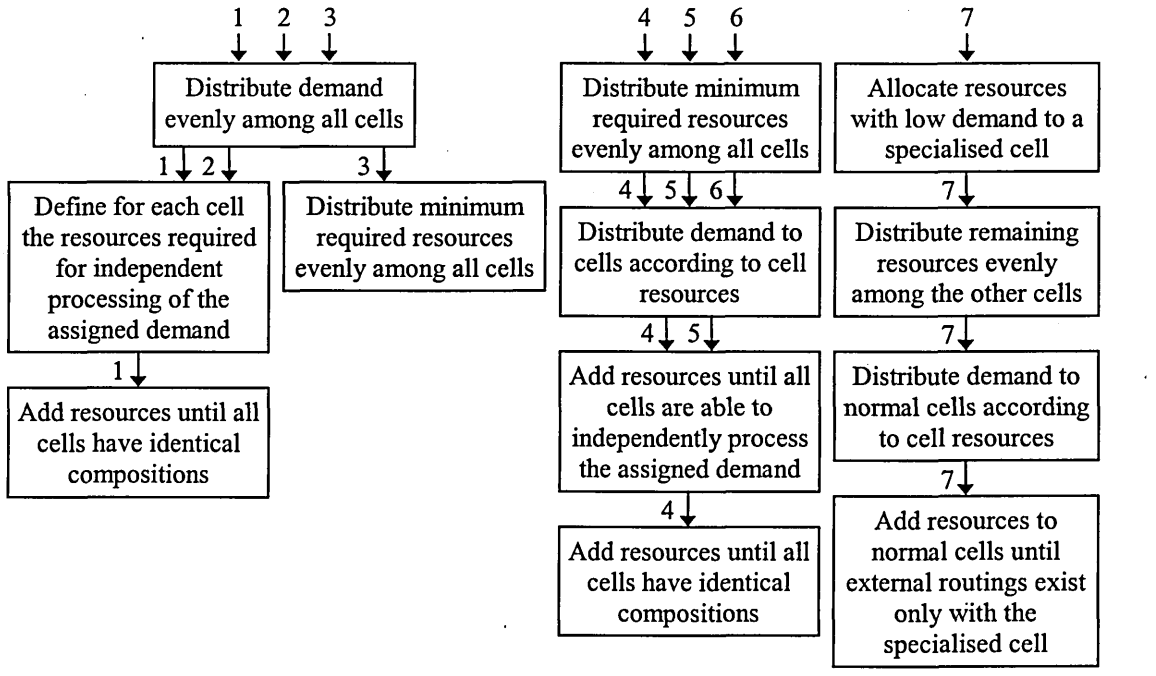



**Figure 5.2** Allocation of machines and products to fractal cells

The third major issue in the design of fractal cells is the level of cell autonomy labelled in figure 1 as product routing. Inter-cell product flows can either be prohibited, unconstrained or restricted to occur only if the required resource is unavailable within a cell. Montreuil *et al.* (1999) argued that the objectives of resource utilisation, flow reduction, and system flexibility are in favour of non-independent fractal cells. Conversely, they noted that the material flows in autonomous cells are easier to control. In the proposed autonomous methods 1, 2, 4 and 5 no product routings between the cells are allowed at all. Methods 3 and 6, on the other hand, allow material flows between the cells, if the local cell has no remaining capability for the required resource. In figure 5.1 this routing approach is called cooperative (free) as opposed to the cooperative routing in method 7 where inter-cell flows are restricted to occur only between a normal cell and the specialised cell. The normal cells route products to the specialised cell when its resources are required. After processing in the specialised cell the product will return to the original cell for completion.

Overall, allowing cooperative cells increases system flexibility and resource utilisation, but would also complicate the material flows. Since internal flows are still given priority, the majority of material flows occur within a cell, which simplifies control. In the case of a machine breakdown, the products can be routed to other cells. Adding extra machines may improve material flows and system flexibility towards increases in demand. With autonomous cells, on the other hand, a machine breakdown can bring a whole cell to a standstill, since cells have to operate independently. Finally, capacity requirements are generally higher in autonomous cells.

The optimal number and size of fractal cells and the arrangement of the machines on the shop floor are design issues that will be discussed briefly. However, in the proposed methods these issues are the result of cell configuration rather than pre-selected design parameters. The number of fractal cells is decided after evaluating the performance of the system simulation using different cell quantities. Montreuil *et al.* (1999) suggested setting the number of fractal cells to equal either the smallest number of machines of one type or to the average number of machines. To achieve an even machine distribution they suggested using an ordered machine list. If identical cells are desired, the number of fractal cells may be selected so that all available machine types can be

equally distributed among all cells without significant growth in capacities. In any of the proposed fractal design methods the resulting cell size needs to be considered when selecting the number of fractal cells. The advantages of a square arrangement of machines in an agile environment were recognised by Askin *et al.* (1996). Venkatadri *et al.* (1997) recommended a value of 5 to 15 machines per cell.

The external layout defining the positions of the cells in relation to each other, referred to as global layout by Venkatadri *et al.* (1997), needs to be considered if cell cooperation is allowed. If the formation of a specialised cell is desired, this cell should then be located in the middle of the shop floor so that the travelling distances from all other cells is minimised. If the specialised machines are placed within normal cells, then cells holding replicates of the same specialised machine type should be placed on opposite sides of the shop floor for optimal distribution and flow distance.

The issue of internal cell layout is significant, because it affects the product flows and travelling distances within the cells where the majority of material flows occur. An identical cell layout can be created for all fractal cells if they have exactly the same internal resources. If the difference in cell composition is only marginal, very similar layouts may be created. The cell layout design is generally more complex in cooperative cells if inter-cell material flows are taken into account as demonstrated by Venkatadri *et al.* (1997). The machine assignment in each cell is mainly influenced by the resulting number of intra-cell flows that would take place, but the direction from which externally routed products flow in and out of the cell is also taken into account during the layout optimisation process.

## 5.4 Developed integrated methodology for fractal shop floor configuration

For the implementation of the proposed fractal cell configuration methods an integrated procedure illustrated in figure 5.3 was developed. The suggested methodology summarises the main characteristics of the proposed layouts and their design processes and indicates the relevant mathematical formulas at each stage. The notation used is listed in the nomenclature.

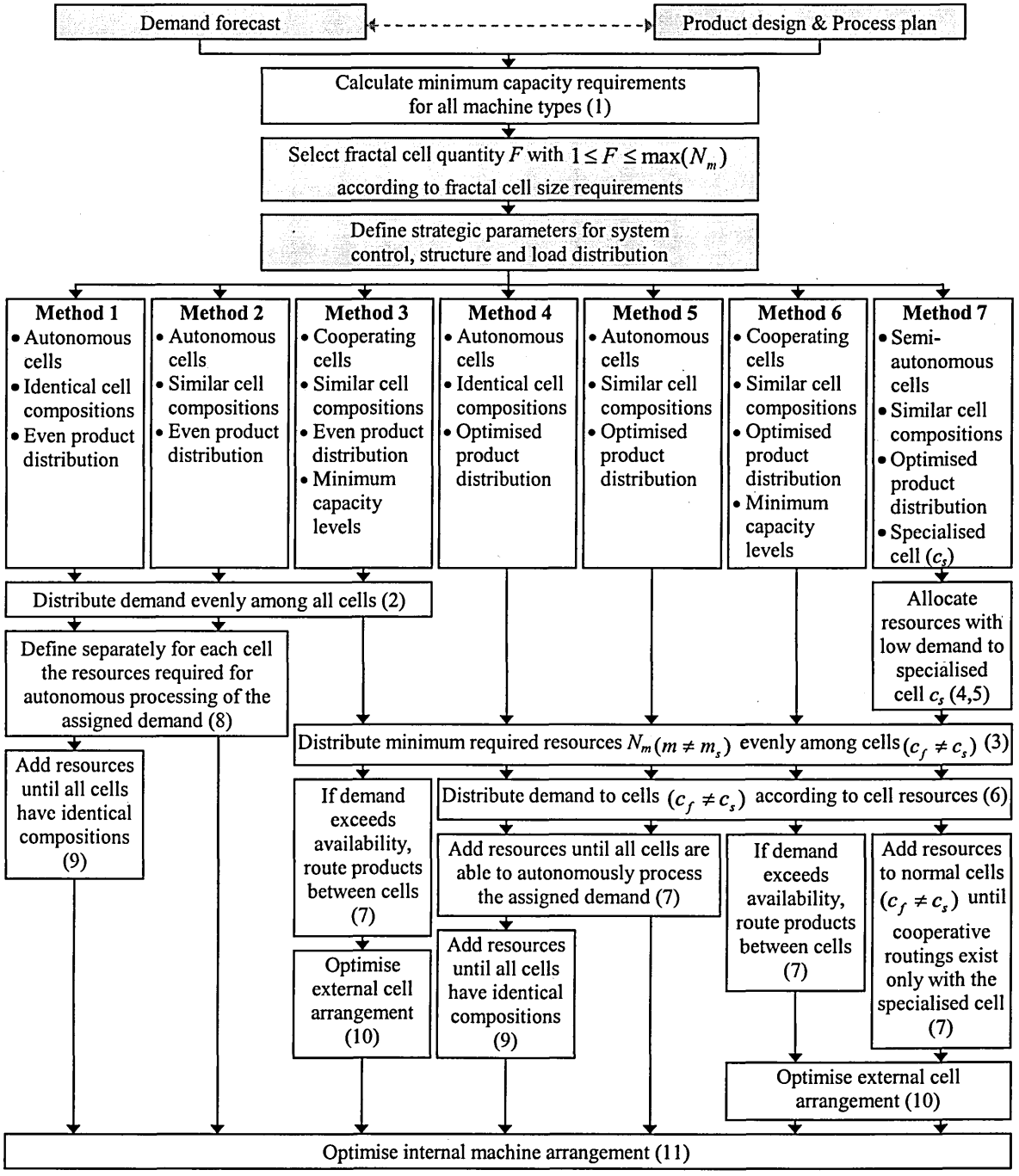



**Figure 5.3** Integrated fractal shop floor configuration procedure

For all fractal configuration methods, the initial stage in the design process is the insertion of the demand and processing data, as illustrated in figure 5.3. As one of the few phases in the procedure that require human input, the stage is highlighted in the diagram. The data is used in the next phase to calculate the minimum machine quantities of each type with formula (1) as suggested by Venkatadri *et al.* (1997):

$$N_m = \left\lceil \frac{\sum_{j=1}^{J} \left(D_j \cdot T_{jm}\right)}{U_m} \right\rceil, \text{ where } m = 1,...,M. \quad (1)$$

Human judgment is also required for the selection of the appropriate fractal cell quantity and the design method. To support these strategic decisions the key design parameters and options have been summarised in table 5.1. Montreuil *et al.* (1999) suggested that the number of fractal cells should equal either the smallest number of machines of any one type or the average number of machines per type. However, the issue is more complicated in the proposed procedure since for most methods the final number of machines is not known at this stage yet. Therefore, it is recommended that all fractal cell quantities within the range of one and the largest number of machines of one type $1 \leq F \leq \max(N_m)$ are evaluated, and that the cell quantity with the best capacity-efficiency ratio is selected. Method 7 is only applicable for fractal cell quantities where the required quantity for at least one machine type is smaller than $F$-1, or $\exists N_m < F - 1$.

The design parameters governing the strategic features in the control, structure and load distribution of the fractal layouts include cell autonomy, system capacity, cell composition and demand allocation to cells. Table 5.1 lists the main options in these categories. The interdependency of the parameters complicates the system planning i.e. the choice made in one category limits the options in others. If autonomous cells are chosen, a duplication of resources is often necessary and cell capacities have to be carefully considered when products are allocated to cells. The seven proposed fractal configuration methods represent different combinations of the design parameters that seem sensible from a business point of view (figure 5.3).

Meeting another design parameter listed in table 5.1, the fractal configuration procedure always aims for a square-arrangement of machines and cells following the recommendation of Askin *et al.* (1996). The design criterion $y \leq x \leq y+1$ directs the shape of both the cells and the shop floor by restricting the difference between the horizontal and vertical dimensions to one machine or cell. Likewise, no similarity is required among the cells in terms of machine arrangement. For every method the layout in each cell is optimised separately for the demand assigned.

Two main approaches amongst the design methods in figure 5.3 can be identified i.e. cell capacities are based on the evenly distributed demand or demand is allocated to cells according to their resources. In methods 1, 2 and 3 the demand $D$ is first distributed evenly among all cells, limiting the quantitative difference between the cells to one unit in terms of both the total demand $D_f$ and any product type $D_{fj}$ when product demand is not divisible by $F$ with formula (2):

$$D_{fj} = \left\lfloor \frac{D_j}{F} \right\rfloor + l_{fj} \tag{2}$$

$$\text{where } l_{fj} = \begin{cases} 1, \text{ when } \sum_{f=1}^{F} l_{fj} < D_j \bmod F \wedge D_f \leq D_i + 1 \text{ for } i = 1,...,F \\ 0 \text{ otherwise} \end{cases} \quad \text{and } j = 1,...,J.$$

Similarly, in methods 3, 4, 5 and 6 the minimum machine quantities $N_m$ are distributed evenly among all cells $c_f$ with:

$$N_{fm} = \left\lfloor \frac{N_m}{F_R} \right\rfloor + l_{fm} \tag{3}$$

$$\text{where } F_R = \begin{cases} F-1, \text{ if } \exists c_s \\ F \quad \text{otherwise} \end{cases}$$

and

$$l_{fm} = \begin{cases} 1, \text{ when } \sum_{f=1}^{F} l_{fm} < N_m \bmod F_R \wedge N_f \leq N_i + 1 \text{ for } i = 1,...,F_R \\ 0 \text{ otherwise} \end{cases} \quad \text{for } m=1,...,M.$$

For method 7, the machine types $m$ for which condition (4) is true are allocated to the specialised cell $c_s$ if the resulting number of machines $N_s$ in the specialised cell is within the limits of condition (5). The remaining machines $m \neq m_s$ are then allocated evenly among the other cells $c_f \neq c_s$ using formula (3).

$$N_m < F-1, \text{ where } m = 1,...,M \tag{4}$$

$$\max(N_f) \geq N_s - 2 \tag{5}$$

To minimise the number of additional machines required in autonomous/ semi-autonomous methods 4, 5 and 7 and to minimise the number of cooperative routings between the cells in method 6, the products are distributed to cells according to their resources. This objective is expressed in (6).

$$\text{Minimise} \sum_{f=1}^{F_R} \sum_{m=1}^{M} \left| \left( \sum_{j=1}^{J} D_{fj} \cdot T_{jm} \right) - \left( N_{fm} \cdot U_m \right) \right| \tag{6}$$

If the demand in methods 4, 5 and 7 at any machine type in any cell exceeds the availability, then the cell capacities are increased accordingly. In the fully cooperative methods 3 and 6 products that can not be processed internally due to a lack of resources are routed between the cells to utilise the available capacity in other cells. This is performed until condition (7) is satisfied for every machine type $m$ and every cell $c_f$. For method 7 the number of non-specialised resources $m \neq m_s$ in regular cells $c_f \neq c_s$ is increased until cooperative routings exist only with the specialised cell.

$$\sum_{j=1}^{J} (D_{fj} \cdot T_{jm}) \leq N_{fm} \cdot U_m \text{, where } m=1,...,M \text{ and } f=1,...,F \tag{7}$$

In methods 1 and 2 the resources for each cell are defined separately with formula (8) according to the processing requirements of the assigned demand. Since the demand was distributed evenly among the cells, similar cell compositions are expected.

$$N_{fm} = \left\lceil \frac{\sum_{j=1}^{J} D_{fj} \cdot T_{jm}}{U_m} \right\rceil \text{, where } m = 1,...,M \text{ and } f = 1,...,F. \tag{8}$$

Finally identical cells are created in methods 1 and 4 by increasing the number of machines of type $m$ in every cell to equal the largest quantity in any cell $\max(N_{fm})$ following formula (9):

$$N_{fm} = \max(N_{fm}) \text{, where } m = 1,...,M \text{ and } f = 1,...,F. \tag{9}$$

Once the number and type of cooperative routings required in methods 3, 6 and 7 are identified following the condition (7), the cells can be arranged on the shop floor so that the total cooperative routing distance $S_{coop}$ over all products is minimised as indicated in formula (10):

$$\text{Minimise} \sum_{d=1}^{D} S_{coop,d} \,. \tag{10}$$

At the final stage of the fractal configuration procedure illustrated in figure 5.3 the machine arrangement is optimised separately in every cell for all methods with the objective to minimise the total internal travelling distance $S_{int}$, given by formula (11):

$$\text{Minimise} \sum_{d=1}^{D} S_{int,fd} \,, \text{ where } f = 1,...,F. \tag{11}$$

The quality of a given global and internal layout for a given product demand and mix is a function of the total material handling distance $S$, with $S = S_{int} + S_{coop}$. The deliverable of the procedure includes the capacity requirements, cell and machine layouts and product allocation lists for the selected fractal cell quantity and design method. A sample layout is illustrated in figure 5.4. The usefulness of the different configurations can be evaluated by comparing their capacity requirements and material travelling distances. While some formulas are directly applicable to case studies, meeting the objectives (6), (10) and (11) requires an iterative and in most situations a heuristic approach. To achieve better results, the heuristic search can be complemented with a tabu-search based procedure as a guiding mechanism towards an optimal solution.

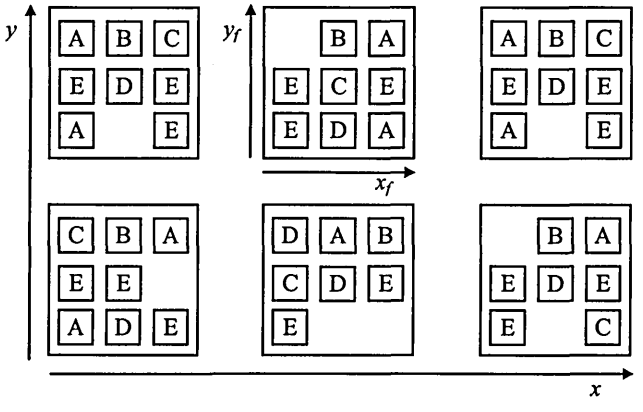



**Figure 5.4** Sample fractal layout

## 5.4 Simulation of the fractal design methodology

A computer program was designed to efficiently carry out all design and optimisation tasks shown in figure 5.3 after human input had been collected. Figure 5.5 depicts a logical flow chart of the fractal configuration for all of the 7 proposed methods. A C++ program, presented in appendix B, has been developed to define cell objects that could be manipulated in terms of global layout, number and types of machines, allocated product demand and mix, and ingress of cooperative material flow. Based on their configurations, the cell objects could in turn evaluate internal and to some extent cooperative material flow distance.

The most fundamental difference between the various configuration methods is the order in which product and machine distribution occur with respect to each other. Methods 1, 2 and 3 first distribute the product demand and mix evenly across the cells and then determine and allocate the machines required to process the assigned products within a given time frame. Conversely, methods 4, 5, 6 and 7 initially distribute merely

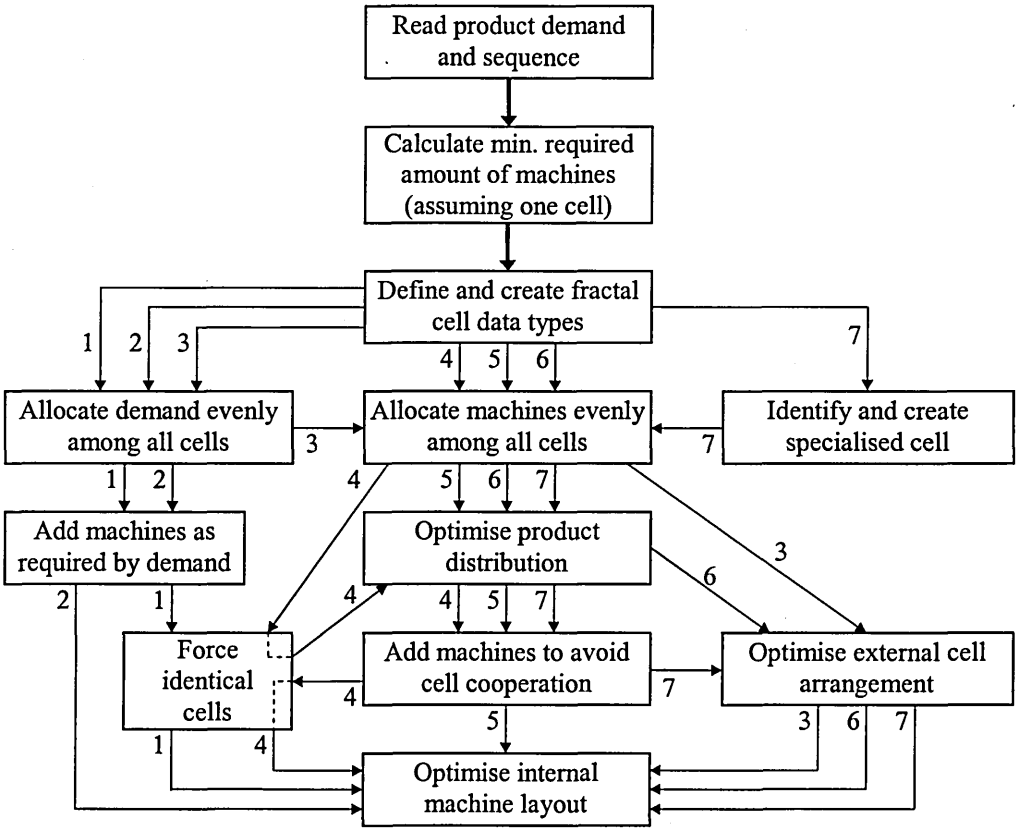



**Figure 5.5** Logical flowchart of fractal layout design procedure

the minimum number of machines across the cells as defined in formula (1). The allocation of product demand and mix is then unconstrained and can be optimised to reduce the number of machines or cooperative material flow. Three heuristic algorithms were needed in the simulation program. One tabu-search algorithm was used for all configuration methods to optimise the internal machine layout within cells. Two simpler heuristic methods iteratively searched a neighbourhood for optimal permutations of external cell layout for methods 3 and 6 and for optimal solutions to the product distribution problem in methods 4 to 7.

Starting with the internal layout problem, the positional two-dimensional matrix of machine types in a cell with size $R_f = x_f \cdot y_f$ is represented by a one-dimensional permutation that is filled row by row, in order to create a neighbourhood for a specific cell layout. The neighbourhood represents all possible permutations that differ from the original layout by exactly one swap move. The formula proposed by Saad and Lassila (2002) for calculating the size of this tabu-search neighbourhood $Z_A$ can be adapted for the tabu-search algorithm on machine layout optimisation as follows:

$$Z_A = \frac{R_f \cdot (R_f - 1)}{2} - \sum_{m=1}^{M} \frac{N_{fm} \cdot (N_{fm} - 1)}{2} \tag{12}$$

If the number of machines $N_f$ in cell $c_f$ is not equal to the size $R_f$ of the rectangular-shaped fractal cell, some positions on the two-dimensional grid will be left empty. The position these empty fields $N_{fe}$ occupy in the $x_f \cdot y_f$ cell matrix influences the flow distances and must therefore be represented in the creation of the permutation. The neighbourhood size for the heuristic search needs to be extended from formula (12) to:

$$Z_A = \frac{R_f \cdot (R_f - 1)}{2} - \sum_{m=1}^{M} \left( \frac{N_{fm} \cdot (N_{fm} - 1)}{2} \right) - \frac{N_{fe} \cdot (N_{fe} - 1)}{2}, \tag{13}$$

where $N_{fe} = R_f - N_{fm}$ for any fractal cell $c_f$.

The proposed tabu-search algorithm assesses the quality of a selected permutation (a given cell layout) by simulating the flow of products assigned to a cell and summing up

the resulting distance $S_{int,f}$ these have travelled. The permutation's neighbourhood is iteratively searched for the non-tabu solution that resulted in the lowest material travelling cost. In addition to machine locations, the material flow simulation takes into account the processing requirements and sequences of allocated products and the availability and capacity levels of the machines. All products are assumed to enter the cell on one side and exit on completion to the opposite direction. These routing distances together with the travelling distances of any cooperatively routed inter-cell product flows are added to provide a more realistic evaluation of the influence a given cell layout has on material handling distances. The external arrangement of the cells is evaluated in a similar manner, but this algorithm did not employ the additional complexity of tabu-search, since the neighbourhood of possible cell positions is significantly smaller.

The combinational optimisation algorithm for product allocation as required for methods 4 to 7 uses a heuristic method similar to tabu search. As experimental results indicated, the maintenance of a tabu list was not required and simple swap search through the neighbourhood of adjacent permutations sufficed. The quality of each permutation is assessed in terms of the number of products that can not be allocated to any cell without exceeding the cell's processing capabilities. Figure 5.6 shows a sample permutation containing all products $D$. The first products in the permutation are allocated to fractal 1 until a product is encountered that can not be processed given the remaining levels of resource availabilities. This mapping of a permutation of products against cell capabilities is continued for all four fractals used in the example. Leftover products to the right of the permutation are those, which the last fractal could not process under the given time constraints.

The heuristic search is executed until changes to the permutation fail to yield any further reduction in leftover products. At this stage the autonomous and semi-autonomous configuration methods 4, 5 and 7 add one machine to increase the capability of one cell depending on where it can make the most difference to leftover quantity. This process iteratively alternates with the combinatorial optimisation process until no leftovers remain and all products have been allocated to capable and autonomous cells. The method places a high burden on computational resources if the number of products is

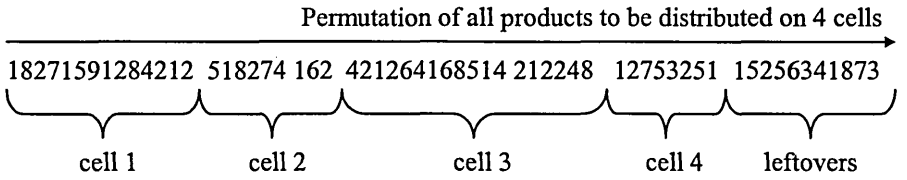



**Figure 5.6** Permutation of products for optimisation of product distribution

large. This corresponds to the number of neighbours that the heuristic search needs to evaluate before committing a combinational swap operation for a given permutation. The neighbourhood size for the optimisation of product distribution can be obtained from:

$$Z_A = \frac{D \cdot (D-1)}{2} - \sum_{j=1}^{J} \frac{D_j \cdot (D_j - 1)}{2} \tag{14}$$

where $D$ is the total number of products, $D = \sum_{j=1}^{J} D_j$.

## 5.5 Results of layout optimisation for a known product mix

The proposed fractal cell configuration procedure was applied to the hypothetical case study introduced in chapter 3 using the developed tabu-search based computer program. The purpose was to validate the effectiveness of the proposed methodology and to evaluate the quality of the different layout design methods in terms of material travelling distances and capacity requirements. Several experiments were conducted for each layout configuration method and a different number of fractal cells. Since the heuristic search could not be expected to yield the optimal solution at the first go, several runs were required, and the results that closest met the layout design objectives presented in the previous section were recorded.

Figure 5.7 illustrates the performance of the proposed heuristic search methodology for an optimal product distribution (method 5). In method 5 and 7 the reduction in leftover products is partly achieved by product distribution and partly by adding further machines at the instance where combinatorial changes in product demand and mix fail

to produce any improvement. As figure 5.7 shows, the process is moving asymptotically towards its objective to distribute all products to capable cells. Towards the end of the procedure, less improvement can be made by changes in the permutation and resource capacities. Figures 5.8 and 5.9 illustrate the machine and demand allocations to cells after the first and last stage of the optimisation process. In figure 5.8 the minimum machine quantities (from formula 1) have been evenly distributed to cells and the demand has been allocated to cells according to these resources following formulas (3) and (6) respectively. However, due to the property of cell autonomy and the limitation that machine type C was presented only in two out of four cells, not all products could be processed with the existing minimal resources. These unallocated products accounted

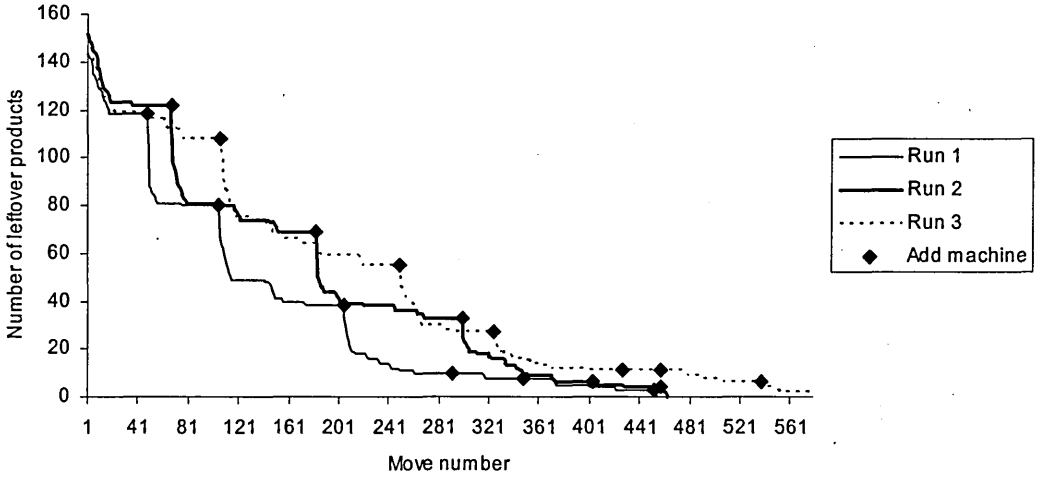



**Figure 5.7** Performance of the heuristic search algorithm

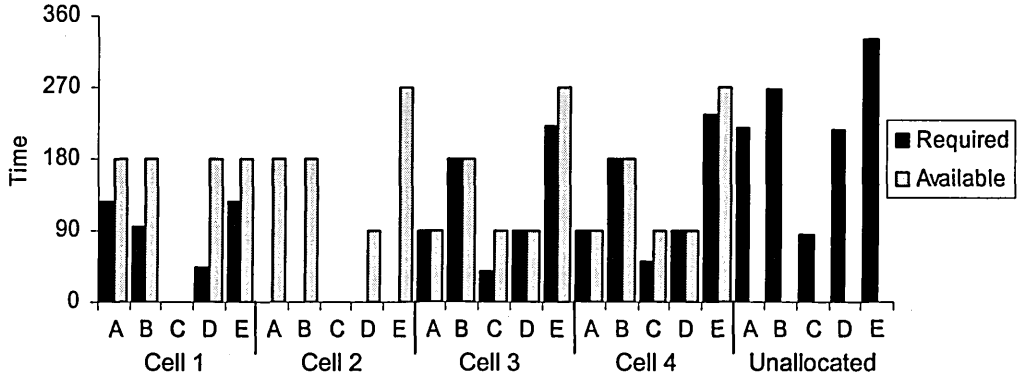



**Figure 5.8** Product allocation optimisation process – first stage

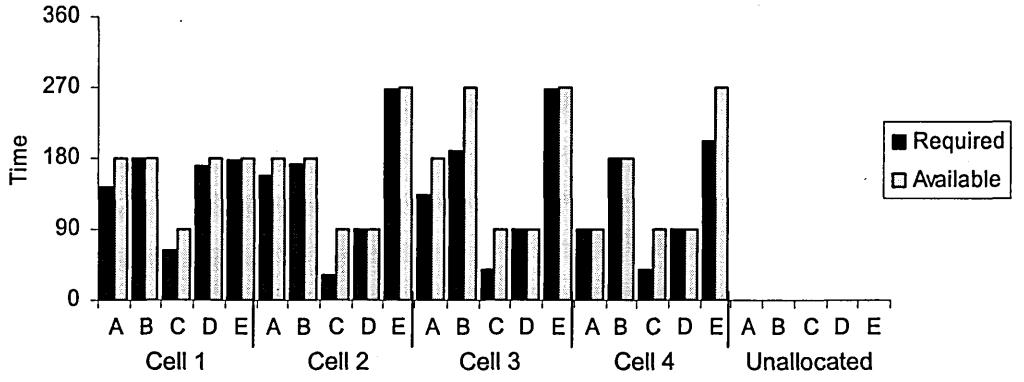



**Figure 5.9** Product allocation optimisation process – final stage

for more than a third of the total demand. It was not possible to allocate many of these leftover products to cells one and two due to the lack of resource C. Using formula (7) duplicate resources needed to be added in conjunction with the optimisation process aiming towards formula (6) to allocate all of the demand $D$ across capable cells. The resulting cell requirements and capabilities are illustrated in figure 5.9, which demonstrates the effectiveness of the procedure to balance the process load with minimum excess capacities.

The external cell and internal machine layouts produced by the heuristic search procedures for all seven methods for a quantity of four fractal cells is shown in figure 5.10. The first observation is that the recommendation of Askin et al. (1996) on the basic square-like shape of shop floor and cell dimensions were met. Further, it is noted that the cell sizes vary among the different design methods chosen (between 32 and 44). For methods 3 and 6 only the minimal amount of machines are placed on the shop floor. However, the full cooperation required between the cells seems to have had a negative impact on the overall product handling distances. These methods may also be more complex to handle from a control point of view. Method 7 provides a trade-off, for which control of material flow is simpler. The design method with the specialised cell also shows the lowest amount of resource duplication above the minimal quantities, but exhibits the heaviest cost in terms of material handling distance. The autonomous design methods on the other hand may be easiest to control but have the largest amount of excess capacity. These conclusions are valid for all fractal cell sizes as can be seen

from figure 5.11, which plots the percentage increase in capacity for all feasible fractal cell sizes in all methods using the data of the hypothetical case study. As a general rule the number of machines in the system grew considerably when the number of fractal cells increased. The excess capacity in methods 1, 2 and 4 exceeded 50% at seven fractal cells. Only the cooperative methods 3 and 6 operated without any extra resources at each fractal cell quantity.

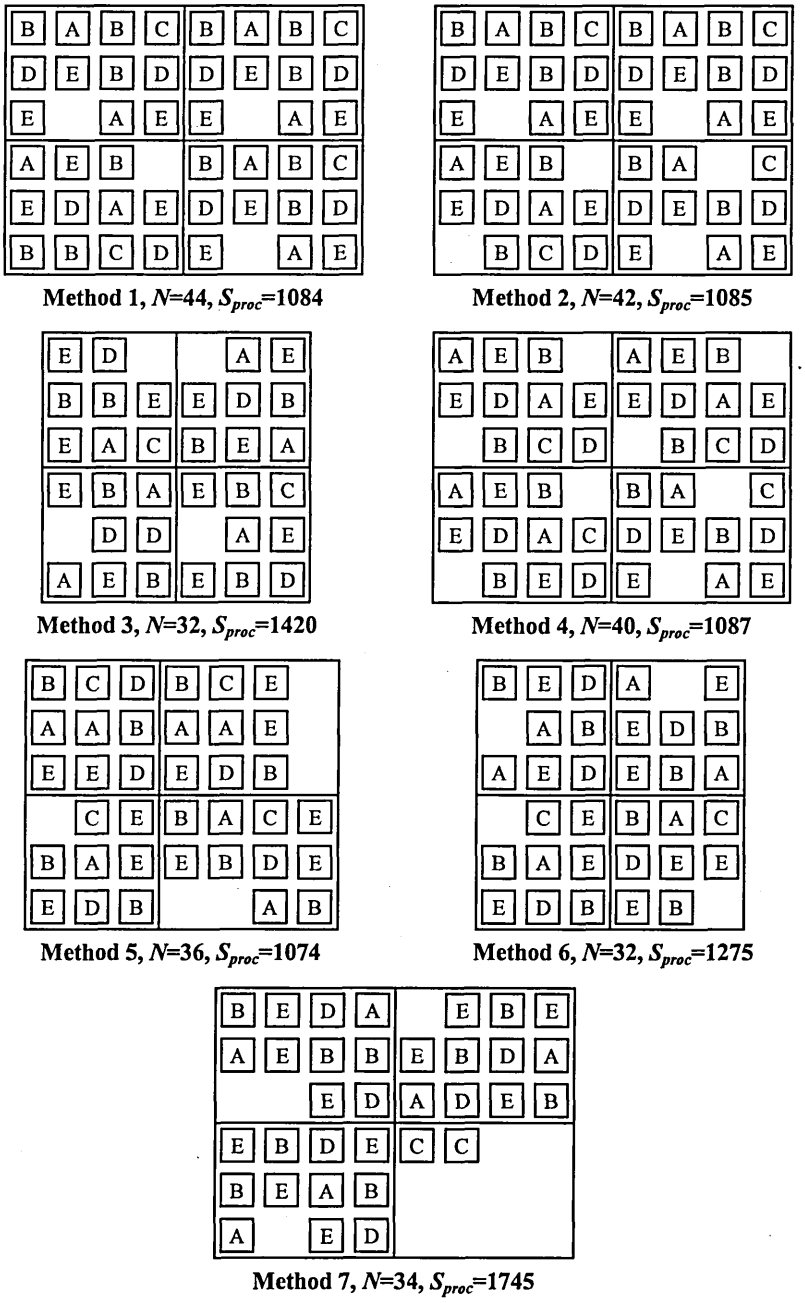



**Figure 5.10** Optimised fractal layouts for four fractal cells

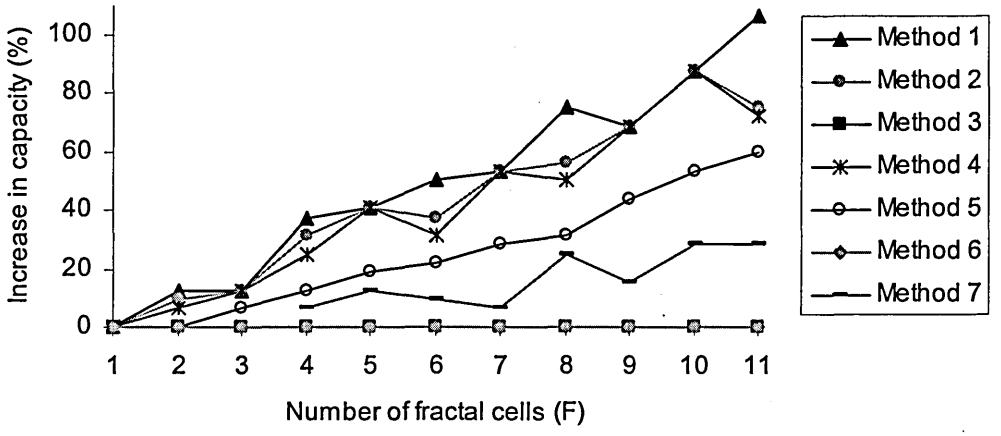



**Figure 5.11** Percentage increase in capacity

Although the difference in the system capacity requirements between the cooperative and autonomous design methods grew rapidly with the quantity of fractal cells, the size of the cells (expressed by the number of machines in each cell) did not change as dramatically. This can be observed from figure 5.12, which plots the average number of machines in a cell for a variety of fractal cell quantities in methods 1 to 7. A gap in fractal cell sizes between the methods establishes at a quantity of four fractal cells. Once the number of fractal cells reaches seven, the average number of machines in each cell for methods 3, 6 and 7 fall below the number of different machine types and continue to decrease reaching as low as 2.9 at the experimental limit of 11 cells. Only 60% of the different machine types are represented in these cells, causing them to rely heavily on cooperation to process a product mix where all products require operations on at least three different machines. Thus, the cell resources do not represent the overall system capacities, and the usefulness of these layouts may be questionable.

Looking closer at the internal cell layout design in figure 5.10, the autonomous cells exhibit a high degree of similarity to each other, even though this was not explicitly aimed for in the optimisation procedure. This apparent similarity can also be found to some extent when comparing the internal cell layouts of different design methods. For method 1 the program produced an almost identical layout to method 2, since both methods have the same distribution of products across cells. Taking cooperation into account for methods 3 and 6, the layout optimisation resulted in very dissimilar internal

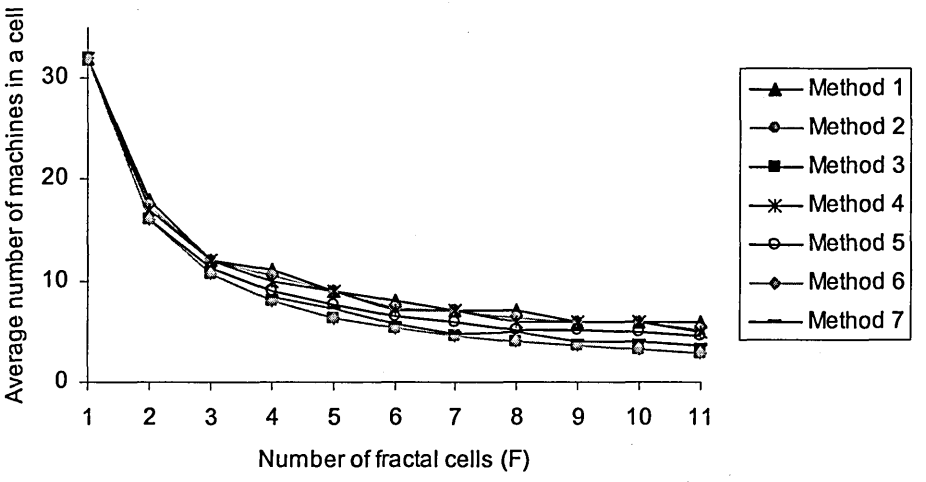



**Figure 5.12** Average number of machines in a cell

cell layouts. This reflects strategic machine placements towards cell borders to facilitate smaller travelling distances for cooperatively routed products.

The actual performance of the layout can be evaluated by comparing the capacity requirements illustrated in figure 5.11 and the travelling distances shown in figure 5.13. For the case study the methods requiring autonomous cells offered the lowest overall flow distances, but required a large amount of machines. Conversely, by allowing cooperative material flows, all products could be processed using only the minimum number of machines as identified with formula (1). However, the material handling distances were consistently larger. The performance of method 7, which allowed cooperative material flows to occur to and from the specialised cell, further supported the claim that a reduction in cell autonomy is generally unfavourable with regard to material travelling distance, even if product distribution, global layout and cell layout are highly optimised.

In the conducted experiments, internal material handling distances $S_{int}$ are considered only for flows that occur inside the cell from the moment a product enters the cell from the left until it exits the cell to the right. Therefore the theoretical minimum travelling distance $min(S_{int,d})$ for every product $d$ equals the width $x_f$ of the cell. The difference in cell sizes is most notable between one and two cell systems where the number of machines per cell is effectively halved. This is the reason for the apparent reduction in

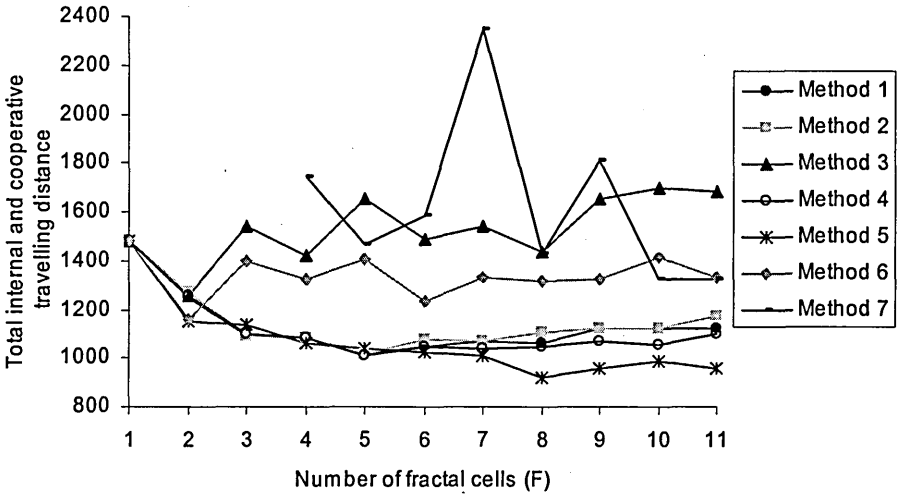



**Figure 5.13** Total internal and cooperative travelling distances

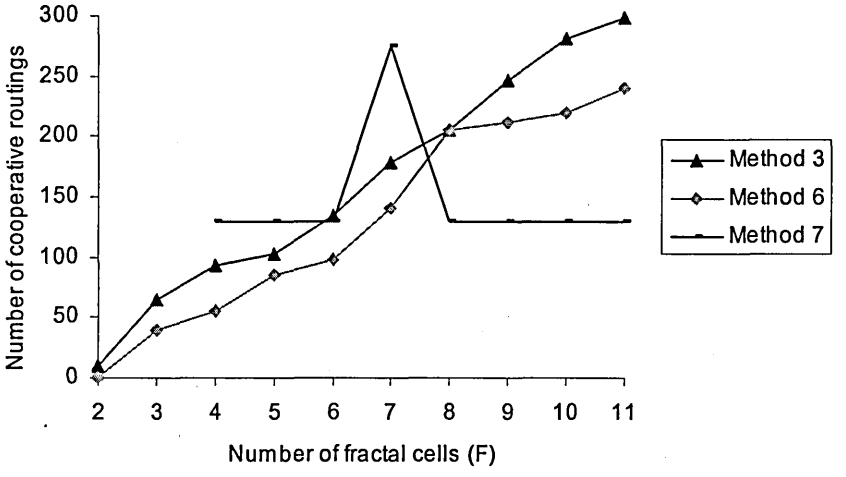



**Figure 5.14** Number of cooperative routings

material travelling distances for all layouts when moving from a single-cell layout to two fractal cells, as shown in figure 5.13. Since a layout with just one cell is not considered to be a fractal system, the single-cell layout is only included for referencing purposes. However, a further increase in the number of fractal cells does not appear to have influenced the overall material travelling distances, even though the capacity requirements grew in most of the configuration methods.

The benefits of the optimisation process can be evaluated from figure 5.14, which indicates on average a 16 percent reduction in the number of cooperative material flows between method 6 with optimised distribution and method 3 without optimised distribution of product demand and mix. The effect of the reduction in cooperative routings shown in figure 5.14 becomes visible in figure 5.13, which demonstrates a consistently better performance of method 6 when compared against method 3.

Table 5.2 provides an excerpt of sample data for the product flow through one cell optimised for design method 2. The relevant cell is illustrated in figure 5.15. The layout also presents a key for converting the machine position data in table 5.2 to the relevant cell locations. Generally, the procedure was observed to arrange the machines in the cells so that any product movements backwards to left are minimised. This is beneficial

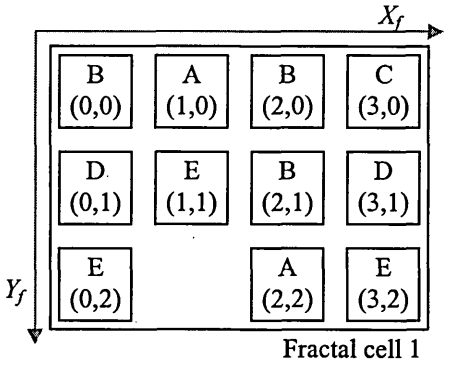



**Figure 5.15** Sample cell dimensions and layout

| Product | Process 1 | Process 2 | Process 3 | Process 4 | Process 5 |
|---|---|---|---|---|---|
| 1.1* | f1:1,0(A)** | f1:1,1(E) | f1:2,1(B) | | |
| 2.1 | f1:2,2(A) | f1:3,0(C) | f1:3,2(E) | | |
| 3.1 | f1:0,2(E) | f1:0,1(D) | f1:0,0(B) | f1:1,0(A) | f1:3,0(C) |
| 4.1 | f1:2,0(B) | f1:3,1(D) | f1:2,2(A) | | |
| 5.1 | f1:3,0(C) | f1:3,1(D) | f1:2,1(B) | f1:2,2(A) | f1:3,2(E) |
| 6.1 | f1:2,1(B) | f1:3,0(C) | f1:3,2(E) | f1:3,1(D) | |
| 7.1 | f1:0,1(D) | f1:1,1(E) | f1:2,1(B) | | |
| 8.1 | f1:1,0(A) | f1:0,0(B) | f1:1,1(E) | f1:0,1(D) | |
| 9.1 | f1:0,2(E) | f1:2,2(A) | f1:2,1(B) | f1:3,0(C) | f1:3,1(D) |
| 10.1 | f1:3,0(C) | f1:3,2(E) | f1:2,2(A) | | |

* product type.replicate ** fractal cell number:*x*,*y*(machine type)

**Table 5.2** Sample product flow

to the travelling distance since all products are required to exit the cell to the right. Therefore, any movement aims to respect the direction of the main material flow.

When the material travelling distances in the optimised fractal layouts are compared to the cells with a random arrangement of machines, a significant reduction is observed. However, the improvement varies notably between the methods and fractal cell quantities as figure 5.16 illustrates. The random data has been averaged over 5000 experiments to reduce statistical variation. Generally, the improvement in material handling distances is more significant the larger the cell size is (i.e. small *F*). This is due to the longer distances between the positions on the opposite sides of the cell in large cell sizes and the susceptibility of larger cells to layout optimisation. In addition, the performance of autonomous cells seems to benefit more from layout optimisation than cooperative cells. One possible explanation to support this finding is the difference in excess capacity as shown in figure 5.12. The cooperative methods 3, 6 and 7 require fewer resources and result in smaller cell sizes, which offer less room for improvement for layout optimisation. On average, layout optimisation improves processing distance by around 20% with a range of 8% to 38% when compared to completely random machine arrangements, with larger cells exhibiting better results.

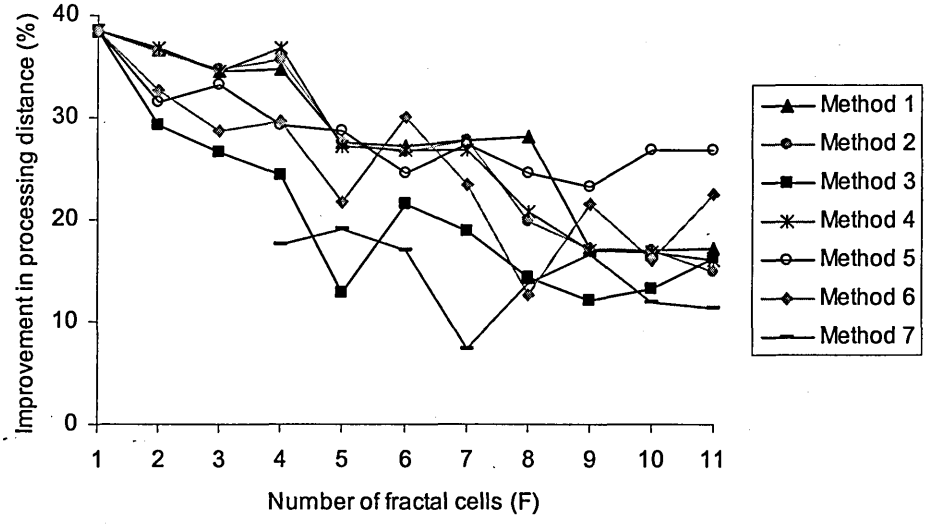



**Figure 5.16** Percentage improvement in processing distances

## 5.6 Flexibility of optimised fractal layouts

Resource layout optimisation yields a notable reduction in material handling costs when compared to the statistically averaged performance of a completely random placement of resources. However, this result is only valid in strict reference to a known set of product demand and mix as pre-defined in the case study. For the investigation on the flexibility of such a layout the resource locations remained fixed according to the arrangements recommended by the previous computational results. A large number of sample orders were then created with the same total demand $D$ but with random demand levels $Dj$ for each product of type $j$. Sample orders that would lead to an excess in capacity levels were discarded. Valid samples were assessed for their degree of similarity to the initial case study, which was expressed as sample difference $G$.

$$G = \frac{\sum_{j=1}^{J} \left| D_{j,sample} - D_{j,initial} \right|}{D} \tag{15}$$

100.000 valid sample orders were created and evaluated for material handling distances for each sample when simulated on cells that were optimised for the initial case study and again when simulated against a background of cells with completely random internal layout. Each case was classified and recorded on a discrete scale of 0 to 199 according to their sample difference multiplied by $D$. The simulation results were averaged over all samples that achieved the same sample difference from the initial state in order to reduce the impact of any statistical variance.

The first experiment was conducted under a system designed for fractal layout method 2 with homogeneous product distribution and similar, but not identical cell composition to reduce the occurrence of resource duplication. Figure 5.17 shows the material handling distances as simulated on a system with two fractal cells. The results of the experiment indicated an increase in material handling distance as the samples deviated further from the case study for which the cell layout was optimised. As figure 5.17 also demonstrates, the handling distance on the optimised cell layout is always lower than on the statistically averaged random cell layout, regardless of the demand levels in the input product mix.

For the generation of valid samples with high sample difference, the constraints under which the computer program generated the random demand levels tended to favour a high demand for products that required a smaller number of processing stages. This trend can be observed from figure 5.18, which illustrates the averaged sum over the number of machines visited by all products to complete the input batch. To make the handling distance $S$ obtained from the samples directly comparable with each other, the results were weighted to eliminate variation caused by differences in the amount $r$ of routings that take place to process the demand $D$.

$$S_w = S \cdot \frac{r_{initial}}{r_{sample}} \tag{16}$$

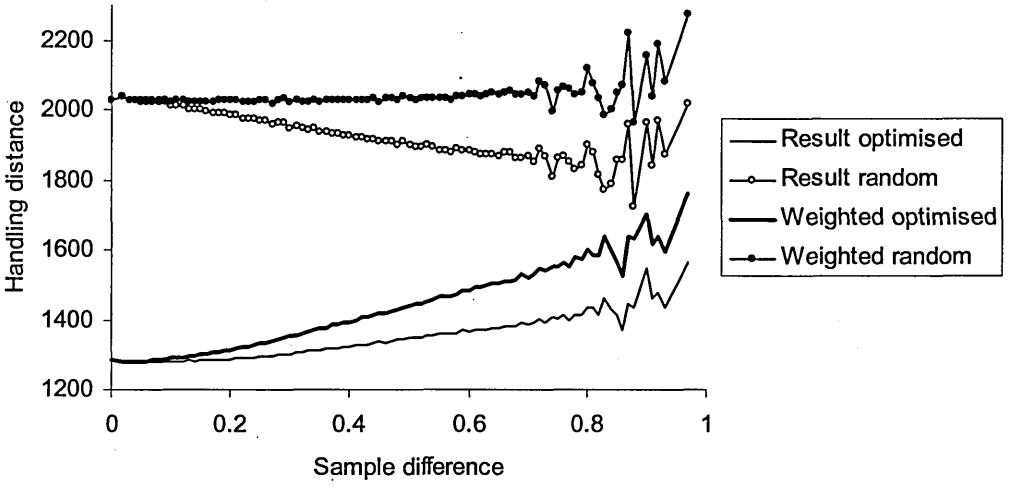



**Figure 5.17** Material handling distance for two fractal cells with similar composition

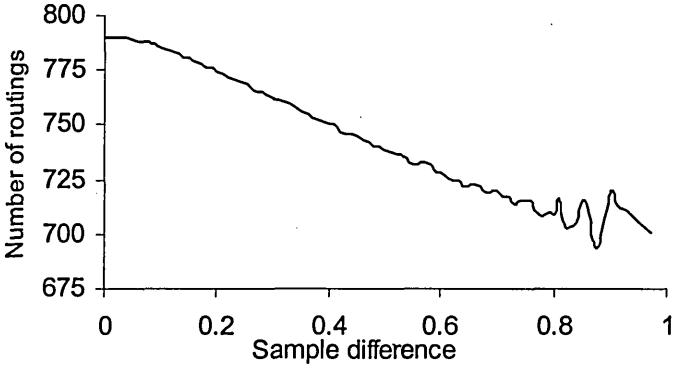



**Figure 5.18** Routings in each sample

The data obtained from weighting the results indicates that material traveling distance $S_w$ over all samples remained constant when simulated on a statistically averaged random cell layout for any comparable random demand levels $Dj$. As shown in figure 5.17 for the optimised layout, $S_w$ increased at a faster rate than $S$ with intensified sample difference, but was still noticeably lower than on the averaged random layout for any sample. Generally, it was observed that the benefits of having an optimised internal cell layout did not deteriorate as quickly as expected when altering the demand levels of input orders for a known mix of products. The plotted graph in figure 5.17 showing the performance of samples on the optimised layout is not entirely linear, suggesting that the flexibility of optimised layouts to changed demand levels is high towards samples that are very similar to the initial case, before results start to deteriorate in strict linear regression.

Since the random samples are organised according to their similarity to the initial set of demand levels, the sample density for each discrete value in sample difference followed a normal distribution. As the amount of data collected for largely dissimilar samples was small, the results of the experiments started to fluctuate around a sample difference of 0.7, because the effect of the variance in random demand levels could not be sufficiently suppressed by averaging over multiple samples. Hence the curves in Figures 5.17, 5.18 and 5.19 oscillate for highly different samples.

When comparing the percentage improvement in material travelling distances for system organisations with different numbers of fractal cells, an almost straight downward trend for organisations with 1 to 4 fractal cells was observed for the selected case study and all its permissible variations in demand levels, as illustrated in figure 5.19. The drop in performance was steeper for systems with 3 and 4 fractal cells than for systems with only 2 cells or no fractal boundary at all. Hence, the layout optimisation for 3 and 4 fractal cells was less flexible towards changes in demand levels than organisations with fewer fractal cells. A fractal system with 2 cells would be expected to exhibit a higher difference between optimised and random layouts for the initial case, because fewer cells with a larger number of resources offer a finer granularity for combinatorial optimisation. This is the reason for the measurements for 5 fractals to start off from a lower percentage improvement between random and optimised layouts.

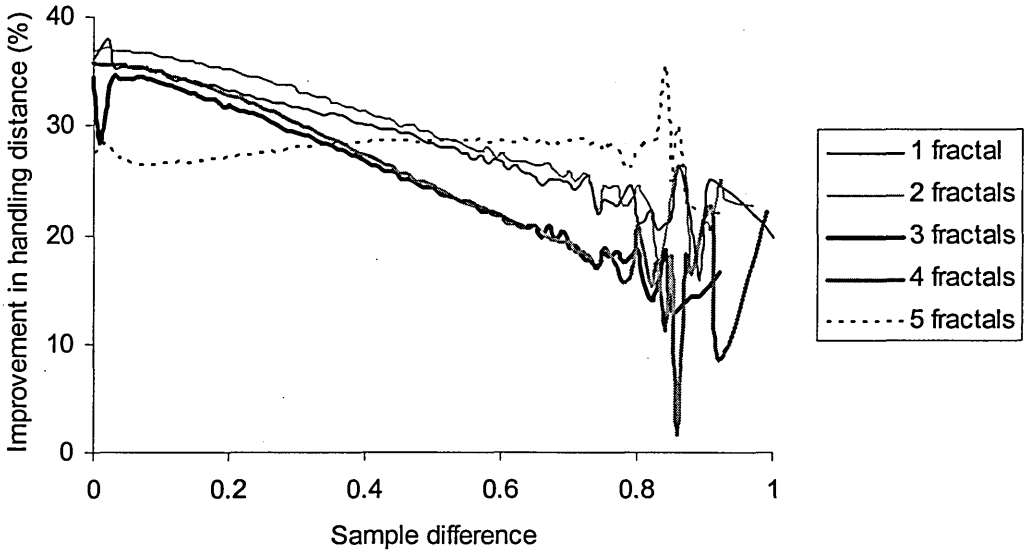



**Figure 5.19** Improvement for varying amounts of fractals with similar compositions

The data produced by simulating a system with 5 fractal cells indicates that cell layout optimisation yielded an improvement in handling distances that was constant at around 28 per cent regardless of the distribution in demand levels in the input product mix. With 5 fractal cells, the number of resources in each cell for the case study was fairly small, 9 machines in a 3x3 square shaped matrix. Therefore, the granularity in combinatorial layout optimisation was so coarse that it did not reflect differences in demand levels, but instead optimised for the known and static processing sequences in the given product mix. For instance, if many product types required processing to commence at machine type A, then heuristic layout optimisation would have located a machine of type A strategically close to the point where products entered the cell, and a difference in demand levels between these products would only be of secondary importance. It was observed from figure 5.19, that the 5-fractal system with the smallest cells could not reflect and model against demand levels at the optimisation process, but instead changed the layout to generally perform well for a given mix of products and the sequence in which they had to be processed. Experiments with fractals that have identical cell compositions (method 1) produced very similar results and merely reinforced the validity of the data.

## 5.7 Conclusions

It was generally possible to reduce material travelling distances by increasing the degree of optimisation of machine layout and product distribution for a specific product demand and mix. Further experiments indicate that arranging the position of resources to meet the requirements for a specific product demand and mix can significantly reduce material travelling distances even if the demand for the different products is unpredictable or chaotic. This observation is valid if the known types of products and their processing sequence remain unchanged. Theoretically the performance of the system with completely dissimilar input parameters, where both demand and mix are altered, should approach the baseline of random cell layouts. In practice, there is reason to believe that although products change over time, their processing sequences may be much more stable. Hence there is merit in optimising the layout of fractal cells for expected input demand and mix. Although the demand for product types may be unpredictable and although products evolve, in practice a noticeable reduction in handling distances of 15 per cent at the very least can be expected when compared to completely random cell layouts. Larger fractal cells benefited more from layout optimisation and a design with smaller more numerous fractal cells resulted in higher inter-cell routings if cooperation was allowed. The benefit of using fractal layouts as opposed to layouts without arbitrary boundaries for a static known product mix was difficult to determine. Where comparable, a small reduction in handling distance was observed when moving from a borderless 32-machine layout to two fractal cells with 16 machines. Fractals do however promise simplified control, the flexibility that all cells are capable to process all product types and some flexibility to changes in product mix. The efficiency of an optimised fractal layout design is a cornerstone for the integrated reference architecture.

# Chapter Six - Biological module development

This chapter suggests a methodology for dynamic self-organisation based distributed scheduling and control in biological manufacturing systems. First, the static and dynamic scheduling approaches and centralised and distributed control techniques are reviewed. Then the exiting biological scheduling methods are summarised. Later in the chapter, a biologically inspired decentralised real-time scheduling methodology based on the continuous communication between machines and transporters is proposed. In addition, three types of dynamic scheduling are identified and the operational procedures are developed. Finally, the proposed methodology is modelled and simulated using Arena simulation software and applied to the hypothetical case study.

## 6.1 Scheduling and control

Scheduling can be defined as "the process of allocating a limited number of resources to a usually greater number of tasks" (Babiceanu *et al.* 2005). It aims to generate a feasible work plan, i.e. what is to be processed where, when and at which order, for the shop floor operations that meets the order due dates and optimises resource utilisation (England 2004). The control function is responsible for the execution of the schedule while taking into consideration the shop floor conditions. The major functions of shop floor control are to ensure resource availability, order dispatching, lot control, managing changes to shop orders, and providing feedback of operational performance (Chase and Aquilano 1992). Traditional hierarchical and sequential production planning and control systems, such as MRP and MRP2, and their limitations were discussed in chapter 2. The need for more flexible and responsive scheduling and control approaches that enable frequent re-scheduling based on the latest system status i.e. changes in production orders and resource availability has been widely recognised in the academic literature, e.g. Brennan and Norrie (2001), Maione and Naso (2001), Alvarez and Diaz (2004) and Babiceanu *et al.* (2005). In addition, Babiceanu *et al.* (2005) noted that determining the optimal production sequences and schedules in a real-world manufacturing environment with resource, precedence and timing constraints is a difficult task.

Dynamic scheduling aims to create more realistic and reactive plans by scheduling operations opportunistically on a real-time basis, contrary to the traditional static way (Kochikar and Narendran 1998). An efficient dynamic scheduler can provide shorter product throughput times, improved machine utilisation and better customer service (Alvarez and Diaz 2004). Kochikar and Narendran (1998) summarised the advantages of dynamic scheduling as better workload balance, improved resource utilisation and greater protection from machine breakdowns. In order to achieve this a dynamic scheduler requires i) a short reaction time, ii) the capability to handle unforeseen and urgent conditions, iii) integration with shop floor control, iv) integration with static scheduling, and v) human participation in the decisions (Alvarez and Diaz 2004). The manner in which the schedule is obtained depends on the structure of the control system. Under centralised scheduling techniques global information is used in order to achieve an optimal schedule, whereas distributed systems rely on local information and negotiations to obtain a flexible and acceptable rather than an optimal schedule (Wang

*et al.* 2004). In distributed control architectures the schedule is normally determined through a bidding process (Brennan 2000). Dynamic scheduling in manufacturing has been considered by many authors, e.g. Selladurai *et al.* (1995), Brennan (2000), Alvarez and Diaz (2004), Lim and Zhang (2004), Babiceanu *et al.* (2005) and Bastos *et al.* (2005).

## 6.2 Scheduling and control in biological manufacturing systems

Scheduling in biological manufacturing systems was considered by Vaario (1996) and Vaario and Ueda (1998a). They proposed a dynamic scheduling approach based on self-organisation. The system used only local information to dynamically adapt to changing conditions on the shop floor without seeking the global system optimum. Vaario and Ueda (1998a) argued that in a turbulent environment global information is not necessary because the global optimum is difficult to calculate and maintain due to i) a lack of information, ii) too many influential factors, iii) unknown optimisation algorithm, or iv) rapid and unexpected changes in information. Hence, they suggested a real-time simulation-based control for shop floor operations in a virtual factory. The products and machines were matched and a schedule was created with a 'self-organisation simulator' that directed factory operations in real-time by continuously calculating the local potential fields of the machines and transporters on the shop floor. An attraction field contained information about machine capabilities or product requirements. The transporters were attracted by the potential field and guided to the right machine.

According to Vaario and Ueda (1997), this bottom-up approach to scheduling lead to a local optimisation with unpredictable global results and enabled dynamic and continuous adaptation to disturbances. Vaario (1996) claimed that the initial results of implementing the self-organisation simulator on a virtual factory were promising. They indicated that the system was capable to adapt to abnormal situations. However, before the virtual factory can be replaced by a real factory, the behaviour of transporters needs to be controlled better to prevent collisions and competition (Vaario and Ueda 1998a). Vaario and Ueda (1998a) also noted that a communication mechanism among the transporters could prevent situations of competition, but might leave the system vulnerable if a transporter failure occurs.

The real-time scheduling methods in both centralised and decentralised systems have been criticised for their lack of a system-wide view and their inability to determine globally optimal solutions (Wang *et al.* 2004). Although the 'one-stage at the time' scheduling approaches might result in longer material flows and lower resource utilisations, their main strength is the ability to rapidly respond to dynamic events and disturbances without complex re-scheduling. The traditional approach in real-time scheduling is to assign a resource for the next stage immediately after processing at a previous stage has been completed according to the conditions at the shop floor at that time. In biological manufacturing systems, on the other hand, the schedule for the next processing stage is not finalised until the product has reached the closest available machine capable of processing the following step (Vaario and Ueda 1998a). This approach ensures that any event that occurs during product transportation is considered. In addition, the size of machine input and output queues and the time they are available for products are limited (Vaario and Ueda 1998a).

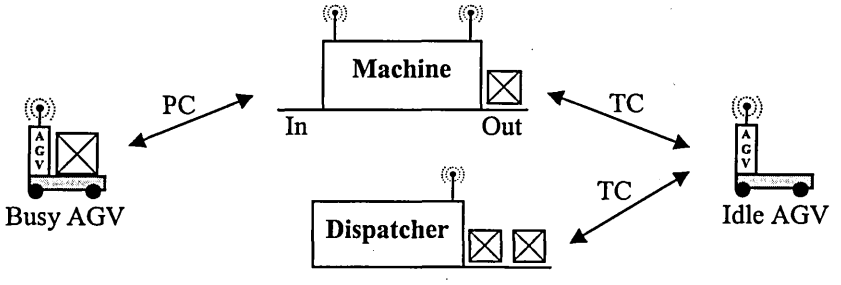
## 6.3 Proposed methodology for dynamic biological scheduling and control

An integrated shop floor scheduling and control mechanism for dynamic self-organisation in biological manufacturing systems was developed based on the work of Vaario and Ueda (1998a). The main differences between the proposed system and the biological scheduling method introduced in Vaario (1996) and Vaario and Ueda (1998a) are the system control, the type of transportation devices used and the presentation of machine and product attractions for matching requirements and capabilities.

The proposed decentralised system removes the need for a centralised scheduler and controller. Scheduling decisions are made in real-time by active components in the system, consisting of one product dispatcher and a number of machine stations and automated guided vehicles (AGVs) that have easy access to all stations on the shop floor. Each of the these components is equipped with a wireless communication device, i.e. wireless local area network (LAN), to transmit and receive data needed for product scheduling, as well as some basic computing device for decision making. The fundamental idea is that machine stations request the transporting service from AGVs whenever products are waiting in the output queue and AGVs request from all machine stations the permission to unload products to their input queue. Capable and available

active components reply to these service requests. The requestor then selects the most suitable device from all replies received according to parameters such as the distance between the components and the estimated time of AGV arrival (ETA) and then signals a message of acceptance to the designated unit. Since the final selection of the service provider is taken by the requestor, there is no negotiation or competition for resources. Figure 6.1 illustrates the process and transport contracts the shop floor elements can form.

Once the processing or transportation contract has been formed, the AGV moves towards the station. Whether the AGV and the station that have agreed on a contract can cancel the agreement during the time it takes for the transporter to reach the station depends on the type of dynamic scheduling used. In table 6.1 three different types of real-time scheduling approaches and the conditions that allow some or all of the parties to break the established contract are listed. In a traditional real-time scheduling situation the transportation and processing contracts for the next stage are formed immediately after processing has completed at the previous station on the basis of the system conditions at that time. No party can break the contract later (referred to as fixed-contract in table 6.1). The other two types of dynamic scheduling contracts listed in the table (flexible-contract and continuous-matching) enable the system to consider and react to events that occur during transporter movement, thus increasing system adaptability. With continuous-matching of contracts both parties continue to look for a better match after the initial contract has been established and can cancel the existing contract as long as the transporter has not arrived at the station. In the flexible-contract model the right to cancel the established contract is granted only to the resource that is 'empty', i.e. empty AGV or machine input control. This stabilises the system while




*Notes:* PC = Process contract. TC = Transport contract.

**Figure 6.1** Process and transport contracts in biological scheduling

| | Dynamic scheduling type | | |
|---|---|---|---|
| **System component** | **Fixed-contract** | **Flexible-contract** | **Continuous-matching** |
| Dispatcher* | Cancellation not allowed | Cancellation not allowed | Cancellation allowed if closer AGV available |
| Machine in-queue** | Cancellation not allowed | Cancellation allowed if product with higher priority available | Cancellation allowed if product with higher priority available |
| Machine out-queue* | Cancellation not allowed | Cancellation not allowed | Cancellation allowed if closer AGV available |
| Idle AGV** | Cancellation not allowed | Cancellation allowed if product with higher priority available | Cancellation allowed if product with higher priority available |
| Busy AGV* | Cancellation not allowed | Cancellation not allowed | Cancellation allowed if closer machine available |

*Notes:* *Active component. **Passive component.

**Table 6.1** Contract cancellation rules for different types of biological scheduling

ensuring that the highest priority product is always processed first. In flexible-contract scheduling the passive component (empty resource) can cancel the contract if a higher product requires processing or transport. In continuous-matching scheduling the same rule is valid for passive components, while the active components (busy resource) can form new contracts if a closer AGV or required machine type becomes available.

The communication on the shop floor can either be multicast (one to many) or unicast (one to one). Multicasting is used by the dispatcher and the station output controllers to request empty transporters to move the products currently in their queues. It is also used by product carrying AGVs to find the closest capable and available machines to process the transported product. The transporters and machines that are available and capable reply directly to the requestor (unicast). The requestor evaluates the replies and selects and acknowledges acceptance to the best resource (unicast). The contract has now been established and the AGV moves towards the station. If cancellations are allowed, the resource will acknowledge to the other party when it has received a better offer (unicast). For a rejected party the process starts again either by sending a request or waiting for a new request (multicast). This process is illustrated in figure 6.2.

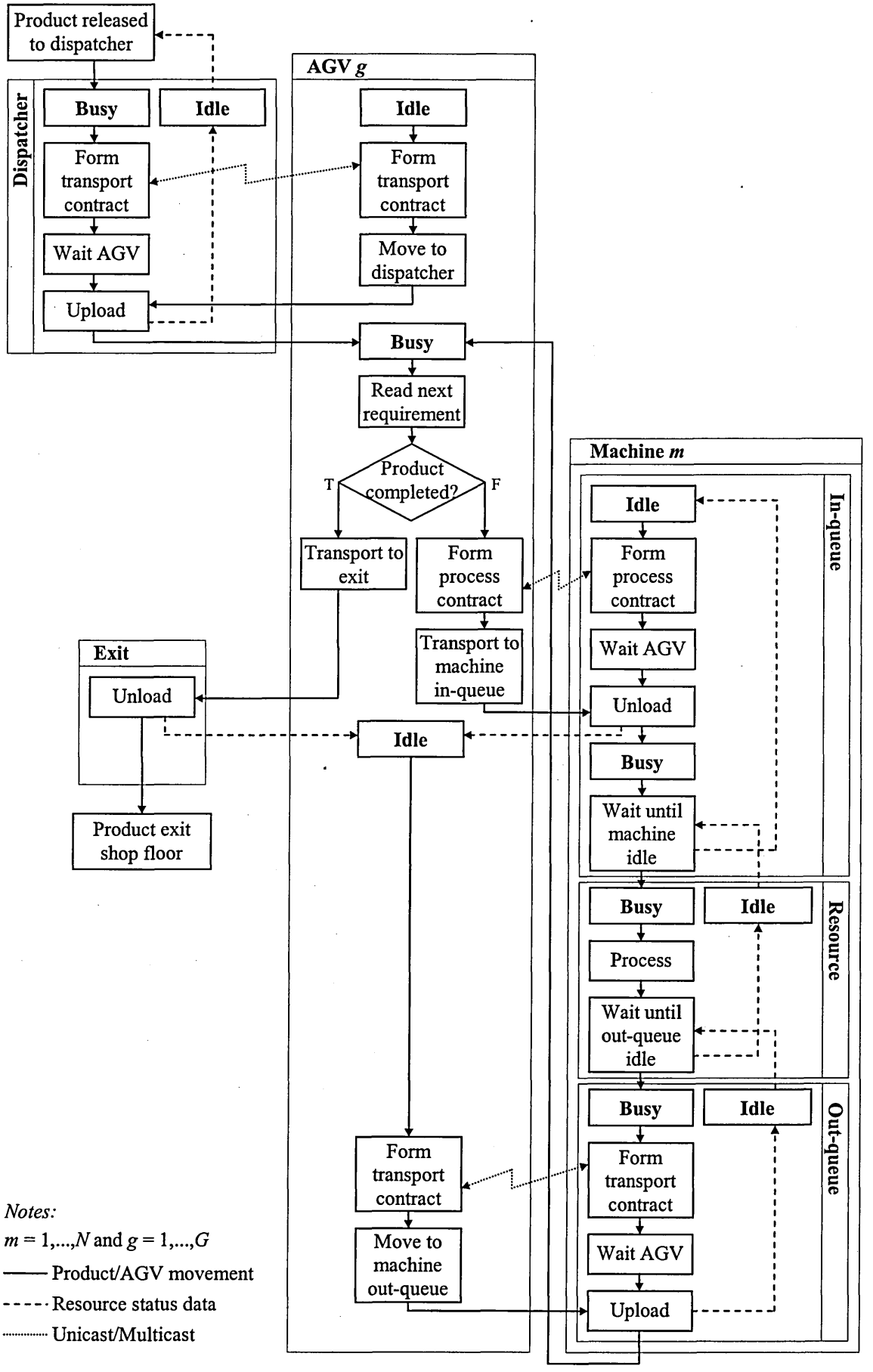
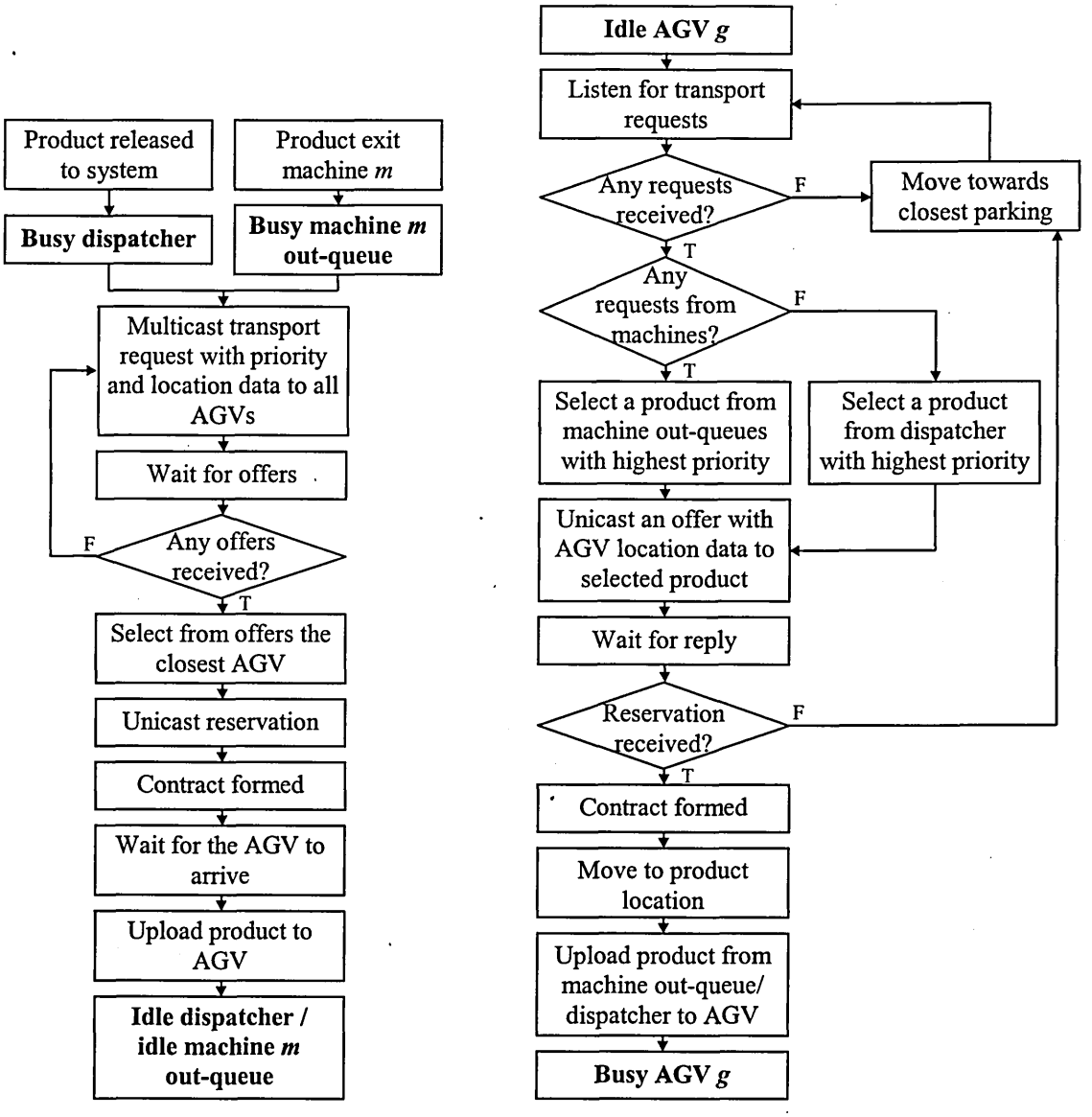



**Figure 6.2** Basic procedure for biological scheduling

The manufacturing instructions needed to complete the whole product (presented in the form of DNA code) are passed between the machine and AGV each time a product is loaded or unloaded. However, only the processing requirements for the next stage (a gene) are multicasted to machines by the AGV when looking for a capable resource. Machines compare the gene data against their own capabilities and either accept or reject the processing request.

The scheduling procedure is simplest for fixed-contracts since no cancellations are allowed. Figures 6.3 and 6.4 illustrate the communication paths between the machine




a) active partner b) passive partner

**Figure 6.3** Transport contracts in fixed-contract scheduling

controllers and AGVs, and the appropriate decisions and actions that are being taken. Figure 6.3 shows the attraction of idle AGVs to machine output queues or dispatcher to pick up the processed products or a new order, while figure 6.4 illustrates the matching of the products carried by the transporters to the closest capable and available machines. Each component has two different states, namely idle and busy, which dynamically change according to product movements in the system. Busy AGVs and busy machine output queues are active elements that initiate the scheduling processes, while idle machine input queues and idle AGVs are passive and only act when requested to do so. Busy machine input queues and idle output queues are inactive (neither active nor passive) until actions from machines change their states to idle or busy respectively.

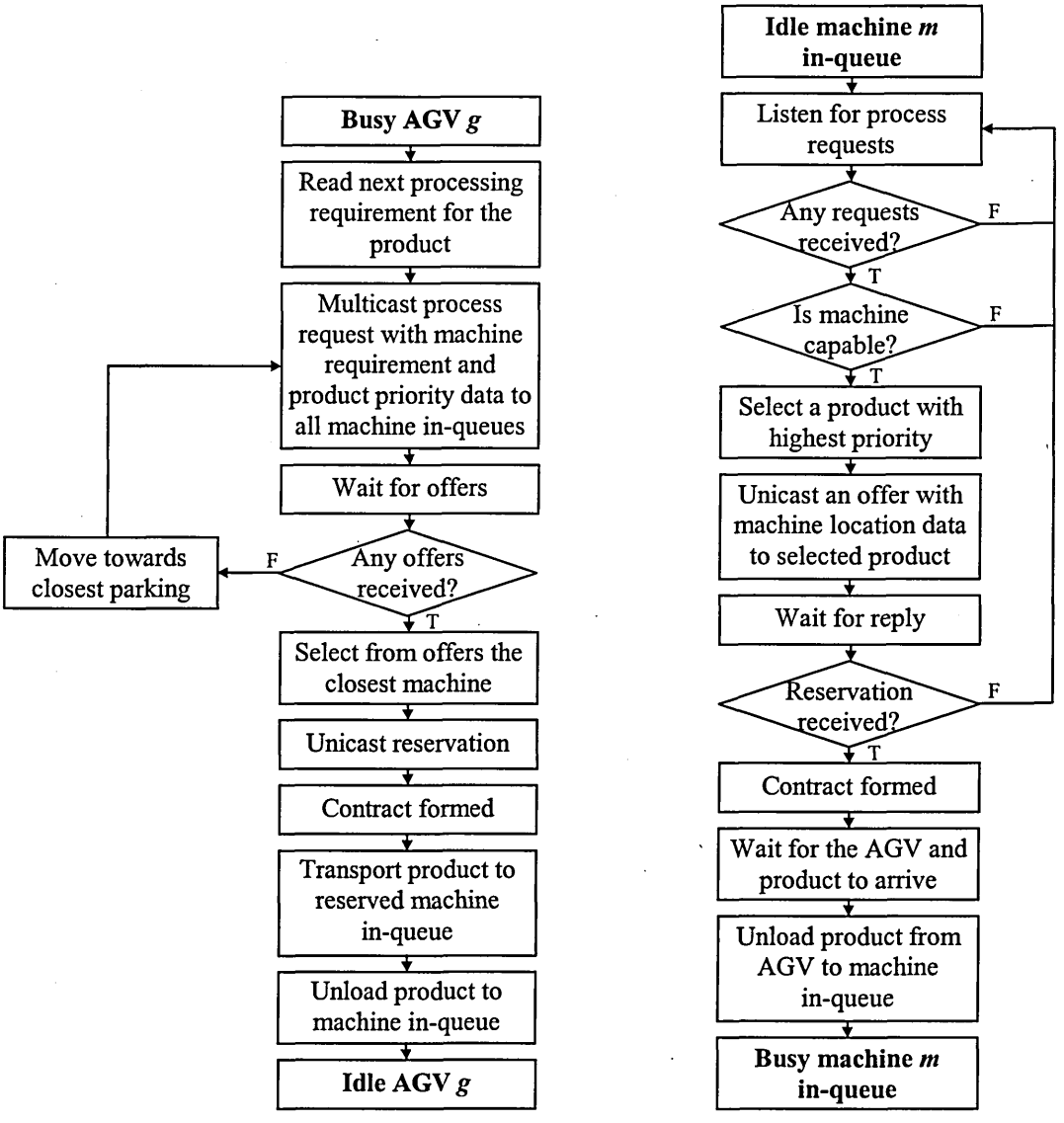



**Figure 6.4** Process contracts in fixed-contract scheduling

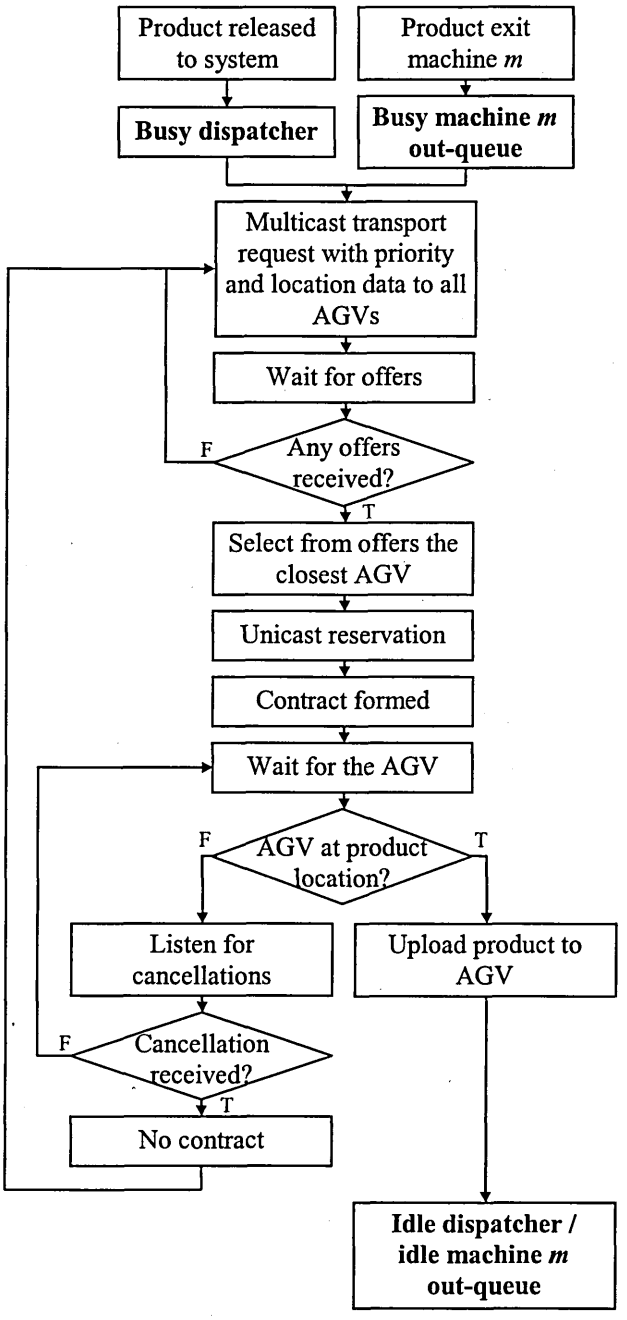



**Figure 6.5** Transport contracts in flexible-contract scheduling (active partner)

The fixed-contract scheduling method is a reasonably stable process compared to the procedures for flexible-contracts (figures 6.5 to 6.8) and continuous-matching (figure 6.9). Flexible-contract and continuous-matching scheduling methods are complicated due to the presence of possible third parties that compete for the same resources. In flexible-contract scheduling, illustrated in figures 6.5 to 6.8, the passive component, which also makes the final decision on the contract, is given the right to cancel the agreement if a product with a higher priority multicasts a request for the resource. Since

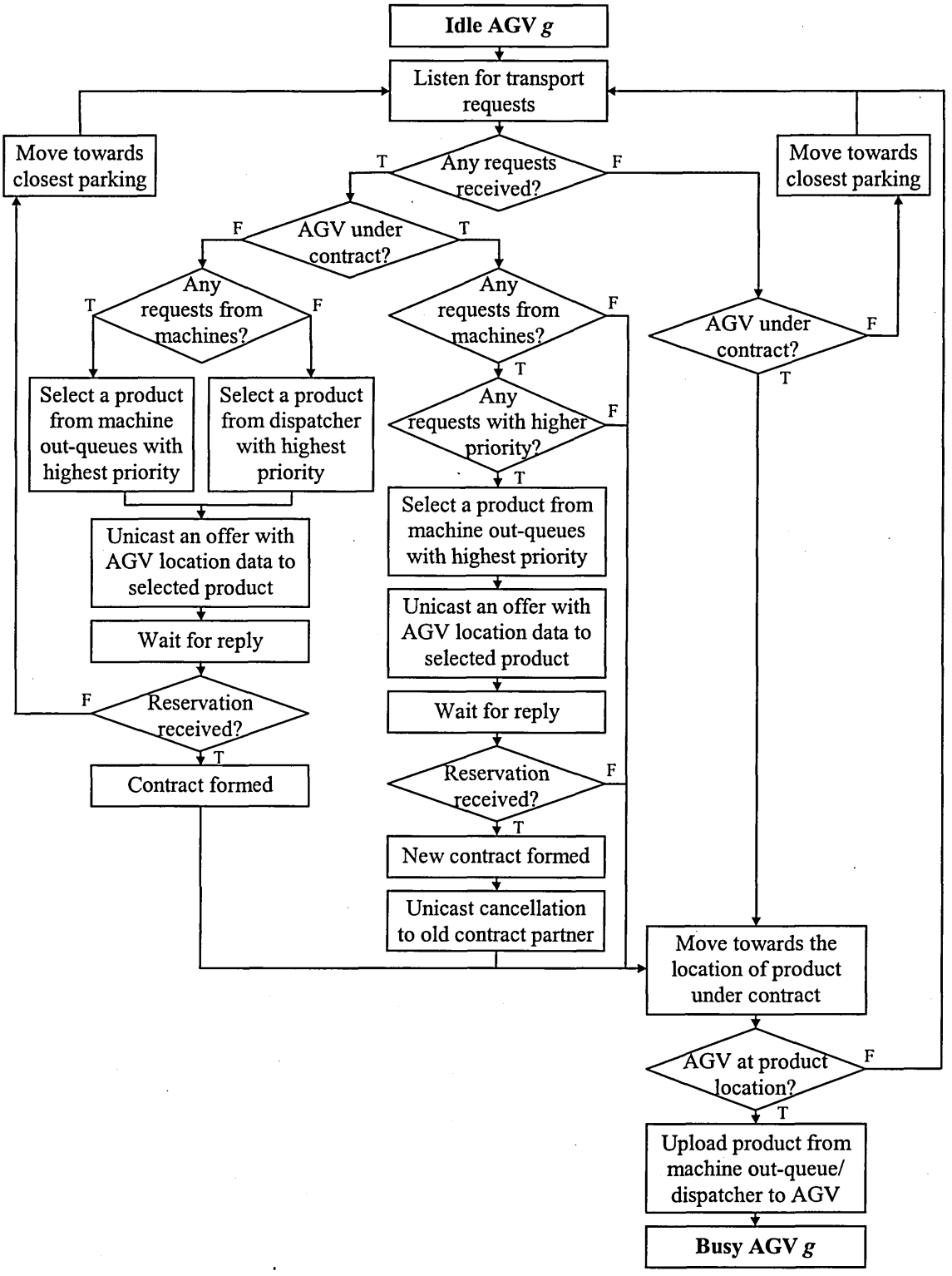



**Figure 6.6** Transport contracts in flexible-contract scheduling (passive partner)

the passive component continues to listen for new multicast requests until the transporter arrives at the station, any new contract can again be cancelled under the

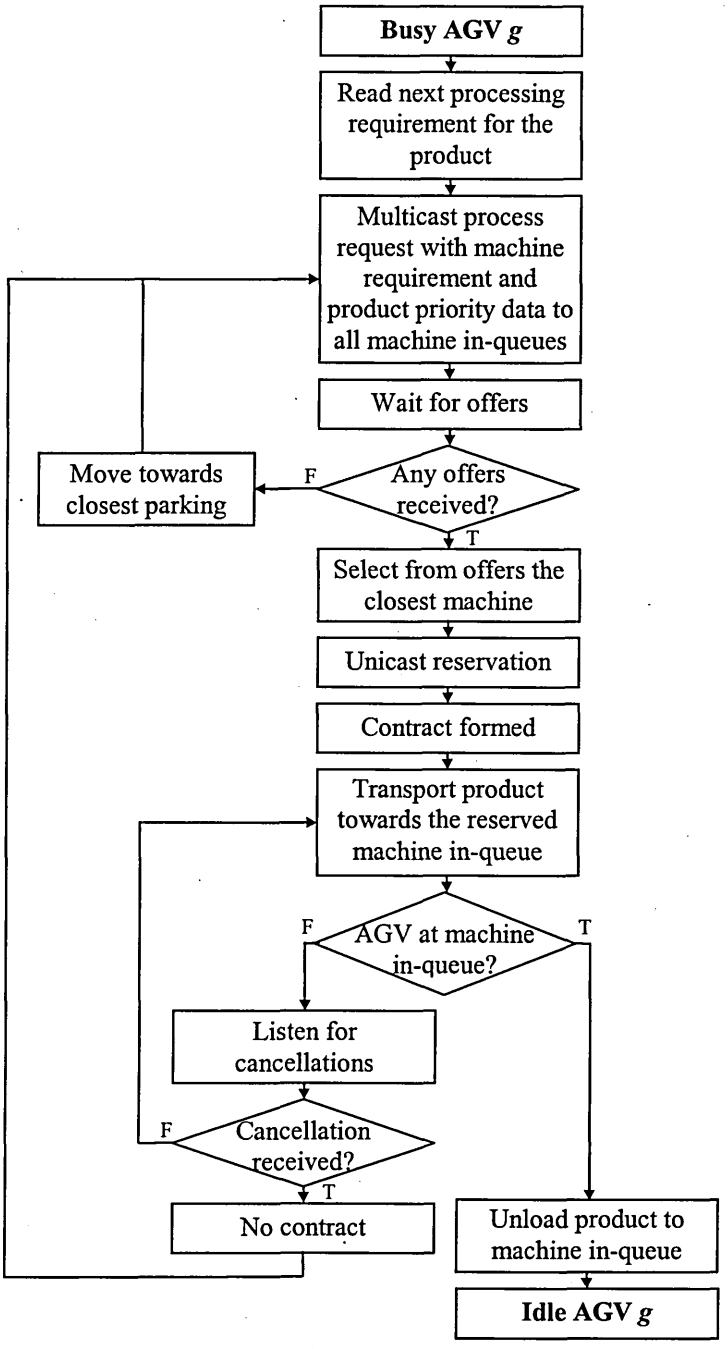



**Figure 6.7** Process contracts in flexible-contract scheduling (active partner)

same conditions. Allowing contract cancellations on the basis of product priority is justified to facilitate due-date completion.

When the right for cancellation in the continuous-matching strategy is extended to active components, cancellations of contracts can occur when a resource at a closer distance is found. This can have either a positive or a negative impact on the overall stability and performance of the system. It enables last minute decisions and the

immediate employment of resources. In theory this should improve resource utilisation and throughput time. However, the continuous cancellation of contracts can leave products with lower priorities circulate around the system, causing AGVs to be reserved for longer than necessary and lead to congestion. To prevent this, the products should be

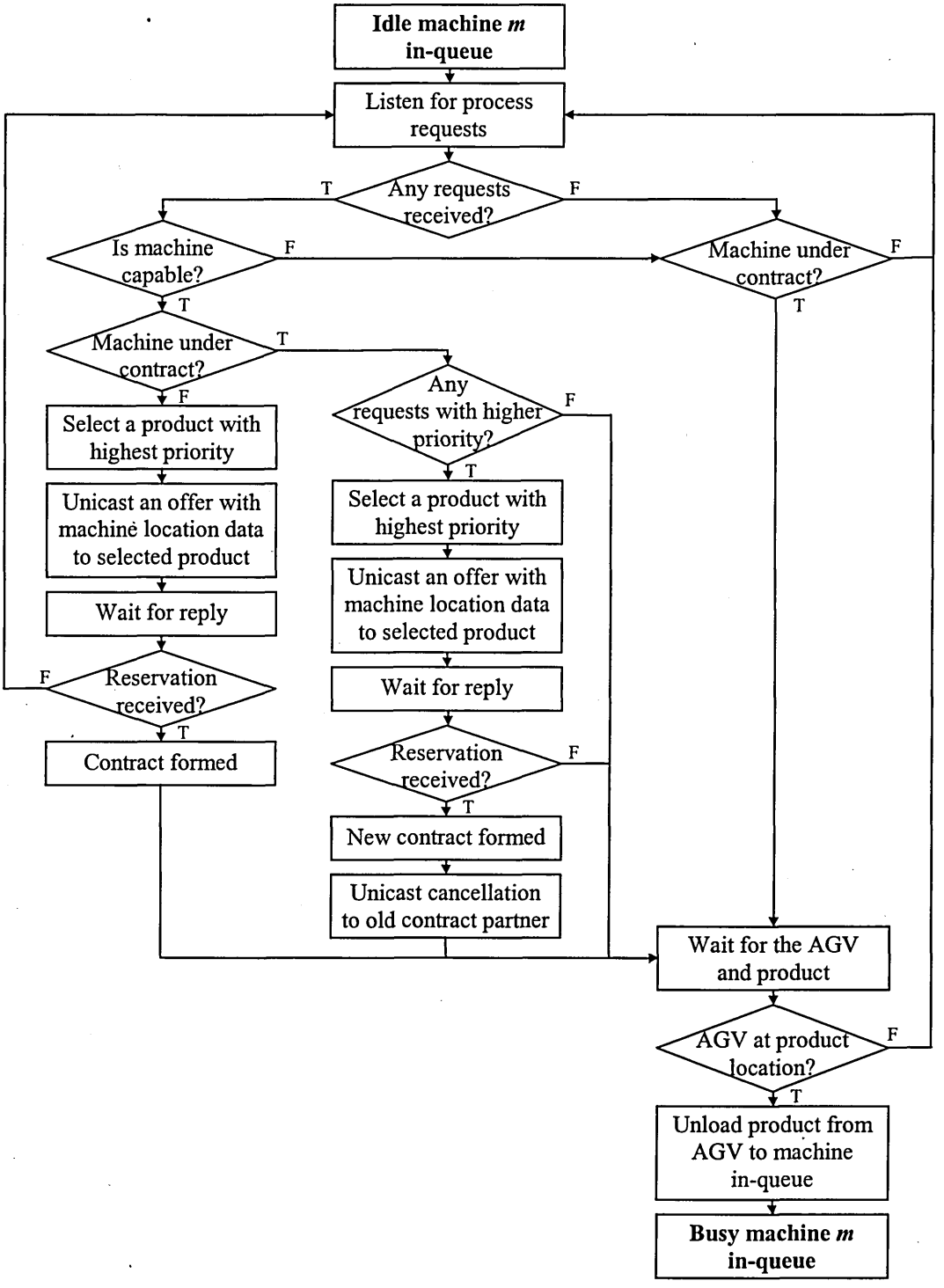



**Figure 6.8** Process contracts in flexible-contract scheduling (passive partner)

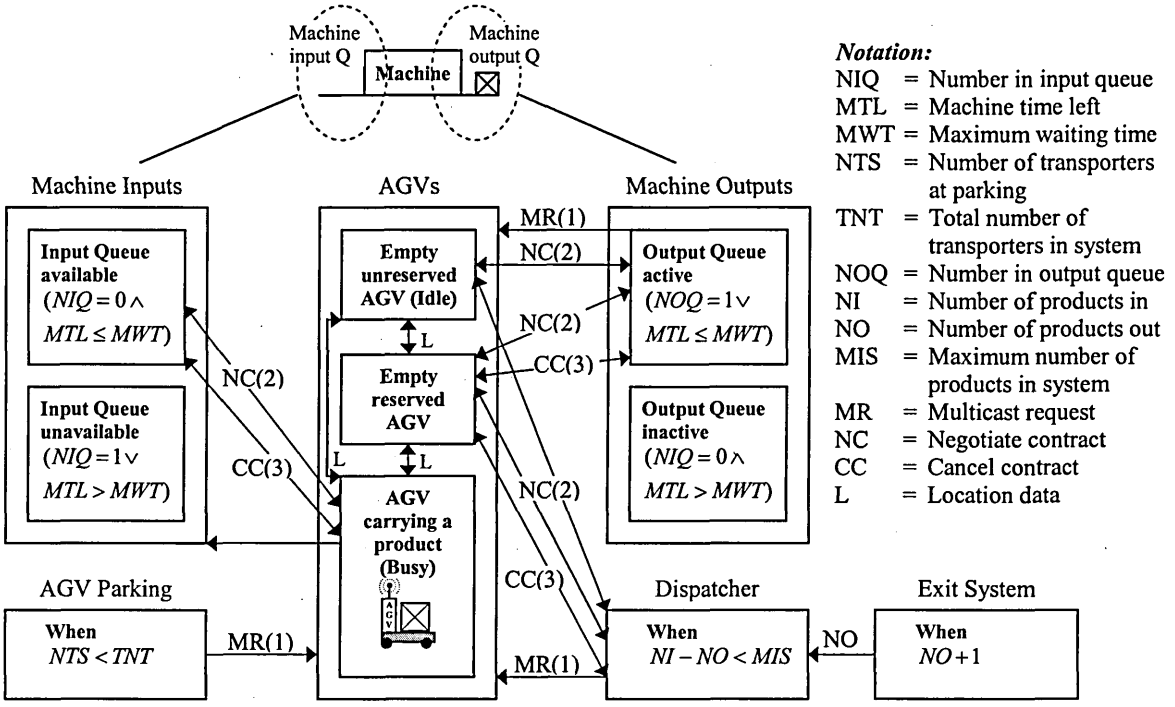



**Figure 6.9** Communication paths in continuous-matching scheduling

sequenced at the dispatcher. Due to the complexity of continuous-matching scheduling the flowcharts for its exact procedures for establishing process and transport contract have not been included. However, figure 6.9 illustrates the communication paths between system elements in continuous-matching scheduling. The figure also presents the conditions that initiate the multicast at each resource. When an AGV in the system is not needed (idle) it moves to the parking area, if the lowest priority attraction from parking becomes the only one affecting the transporter.

## 6.4 Simulation of biological scheduling methodology

For evaluation purposes the proposed dynamic biological scheduling and control methodology for fixed-contract scheduling was implemented using Arena simulation software. The model and layout presented in figure 6.10 was developed using the data from the hypothetical case study introduced in chapter 3. The program code is omitted due to its size. To transport products on the shop floor, the system uses AGVs together with a complex network of unidirectional travelling paths that provide transporters with short-distance access to any machine on the shop floor, illustrated in figure 6.11. All paths are unidirectional with two types of intersections. At major intersections multiple

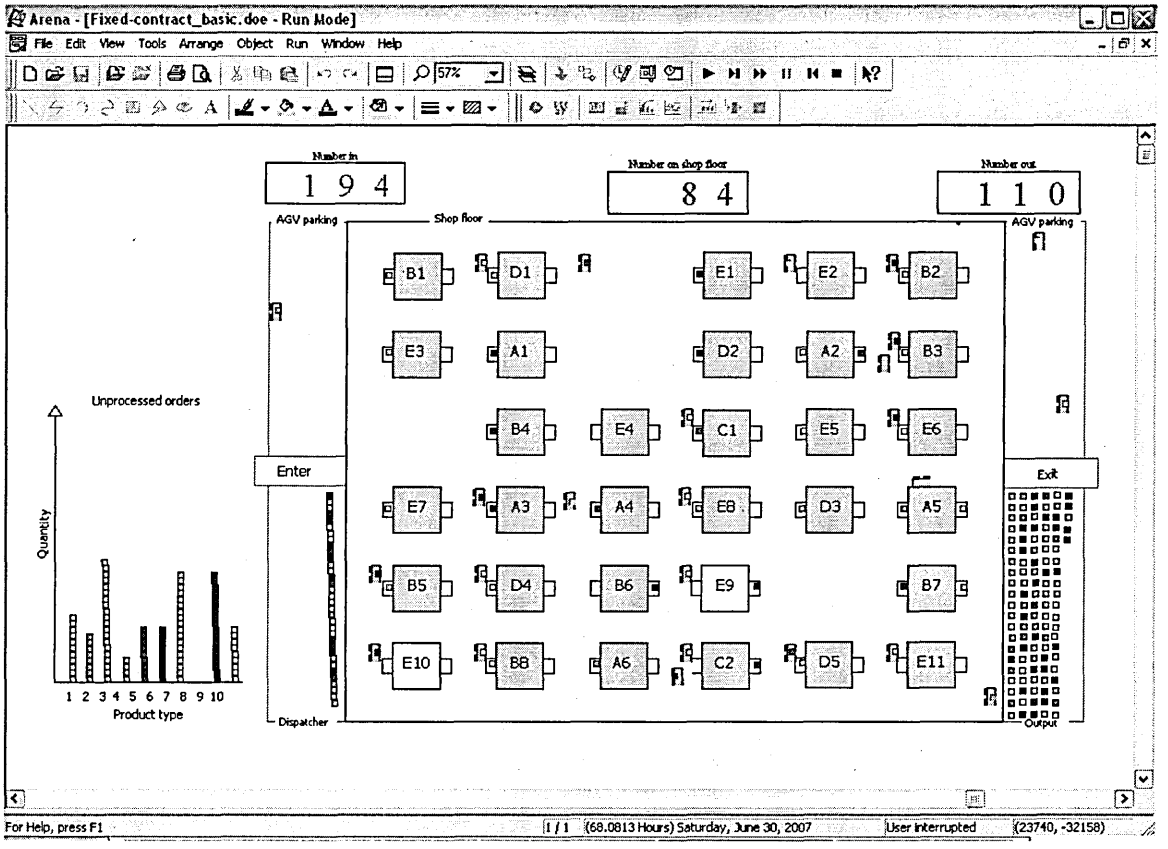



**Figure 6.10** Snapshot of the biological simulation model

travelling paths converge and AGVs can change their direction. Minor intersections are used to load and unload products at machines. Each transporter can carry only one product at the time. Similarly, each machine input and output queue can hold only one product. This can cause starvation and blockage on the machines if products are not moved immediately according to the demand and supply. Therefore, the number of AGVs on the shop floor can be a critical factor for the performance of the system.

The AGVs decide their direction of travel after establishing a contract with a machine based on the priority status of the product in question and/or their current location relative to the destination. Since the AGVs can travel only along the defined routes, a simple control logic together with a communication device enables each AGV to determine its location on the shop floor, decide the travelling direction and prevent collisions by exchanging position data with other transporters. The AGVs detect any possible deadlocks in the system by comparing the current and next positions of the all AGVs on the shop floor. If such a conflict is predicted, the AGV that caused the

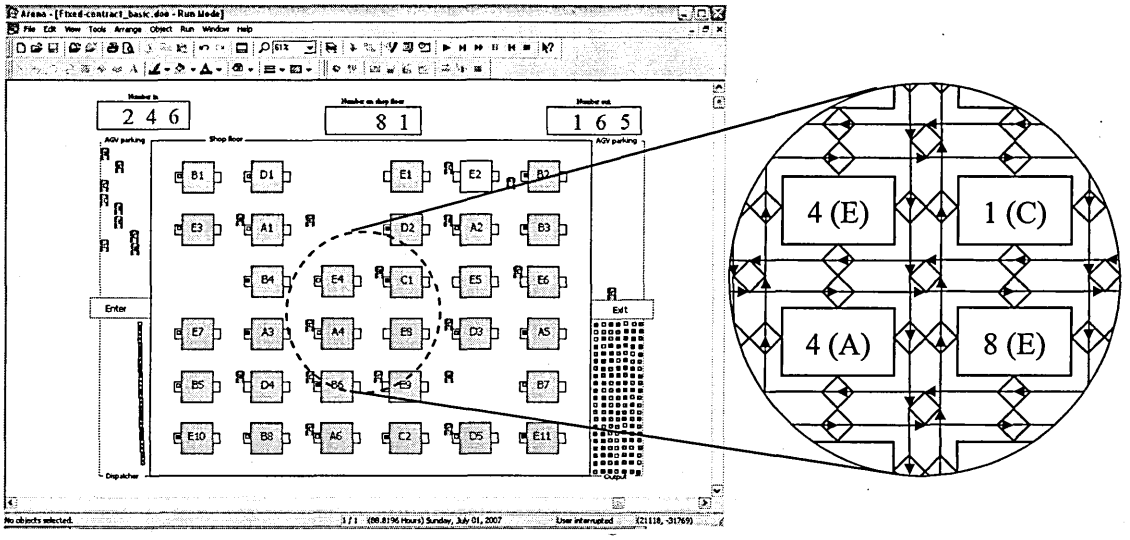



**Figure 6.11** Transporter travelling network

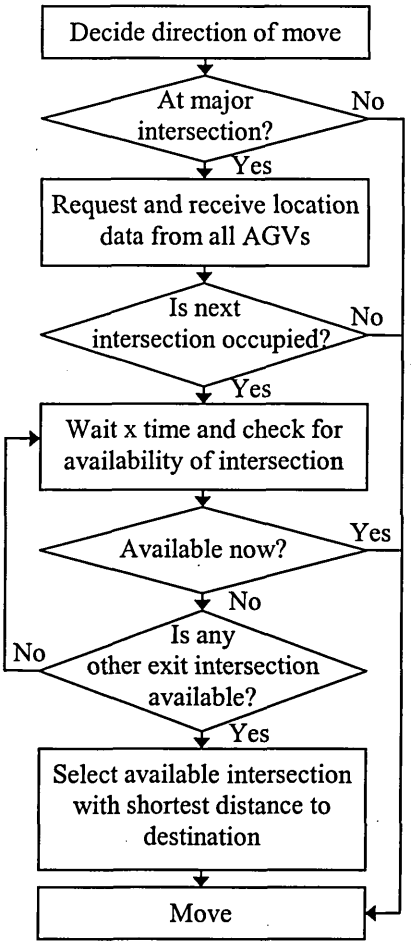



**Figure 6.12** Deadlock avoidance mechanism for transporters

gridlock changes its direction of travel to break the deadlock. The deadlock problem in AGV systems was discussed by Wu and Zeng (2002) who concluded that the only way of avoiding deadlocks when AGVs are used, is to ensure that the circular wait condition never occurs. A mechanism illustrated in figure 6.12 was developed to prevent AGVs from entering into circular-wait gridlocks in the proposed system. In this procedure, transporters are not allowed to move from one intersection to the next if this next zone is already occupied. If this situation persists over a pre-defined period of time, the AGV will examine the availability of zones in the other three general directions of movement and will select from these the route which results in the shortest distance to the AGV's destination.

## 6.5 Computational results and discussion

The developed system considers several critical parameters that significantly influence its performance. Among others these include the number of transporters in the system as discussed earlier as well as the speed of AGVs. From the order arrival point of view, the quantity and mix of products on the shop floor influence both, AGV and machine utilisation and product flow. If the number of products is too high machines and transporters may become blocked, while too low a number causes starvation and low resource utilisation. In the proposed model, the number of products in the system was controlled by prioritising any products at machine output queues over the newly arrived products at the dispatcher. In addition, the product mix on the shop floor at any given time should be a combination of different orders in proportion to the total demand, in order to ensure a better balance in terms of work load between resources.

As described an idle AGV transports a product from a dispatcher to the shop floor only if there are no products in the out-queues of any machine. Hence, the number of products in the system varies according to shop floor conditions. This explains the growth of the lead time in figure 6.14 when the total throughput time reduces, as illustrated in figure 6.13. In general, systems with faster AGVs had better throughput times, lead times and machine utilisations as illustrated in figures 6.13 to 6.15. Similarly, an increase in the number of AGVs improved throughput time and utilisation, but slightly increased lead times as explained earlier. In overall, the dynamic scheduling

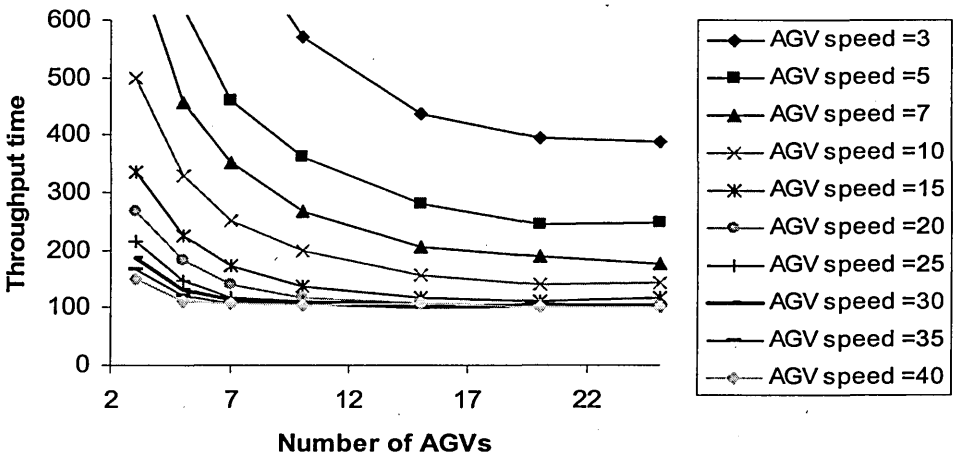



**Figure 6.13** Throughput time for biological scheduling

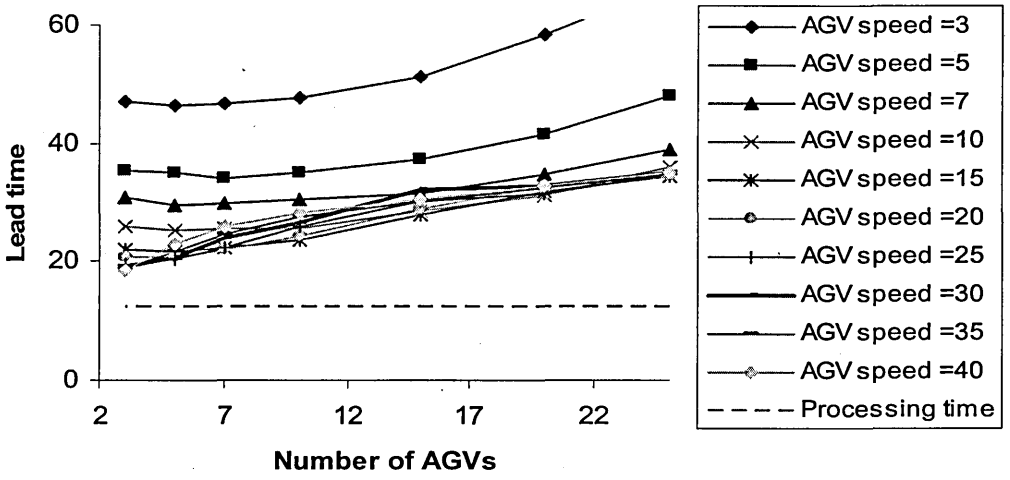



**Figure 6.14** Lead time for biological scheduling

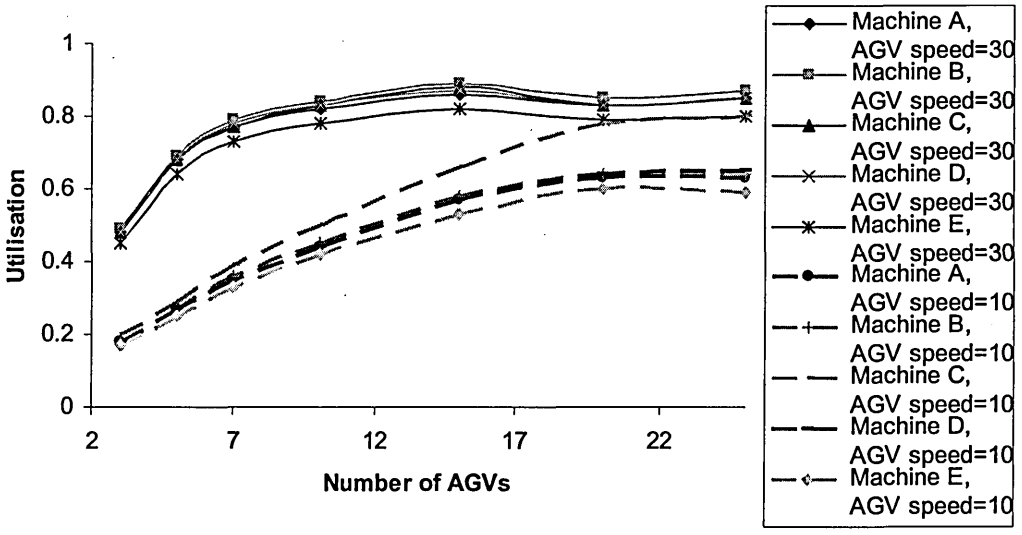



**Figure 6.15** Machine utilisation for biological scheduling

methodology performed very well and the system was able operate close to the due date target of 100 time units with reasonable lead times.

## 6.6 Conclusions

The aim of this study was not to develop an optimal schedule, but to enable a flexible and responsive reaction to any internal or external disturbances on the shop floor. The inherent dynamic nature of the system that evaluates the situation on the shop floor at any given time achieves the required responsiveness to these changes in product demand and mix and any machine breakdowns. This is due to continuous communication between the elements in the system, which enables all parties that form a contract to request a new resource if the contact is lost. No competition for resources occurs, since transporter movements are based on the contracts. The disadvantages of this concept include the restricted ability of movement of the AGVs and the complicated network of travelling paths required that increase the likelihood of the system deadlocks and the complexity for deadlock resolution and avoidance.

# Chapter Seven -

# Developed integrated reference architecture

This chapter develops an integrated reference architecture for the configuration, planning and control of the 21st century manufacturing systems. First, a framework for the integration of fractal, biological and responsive concepts is developed based on the conducted research into the fractal layouts, biological scheduling and control, and resource elements. Next, the detailed procedures for the conceptual interactions are formulated. After that the proposed framework is generalised to formulate the universally applicable reference architecture for shop floor operations. Then the proposed architecture is modelled and simulated using Arena simulation software. Finally, the architecture is evaluated by applying it to the hypothetical case study.

## 7.1 Generalised reference architecture

The framework illustrated in figure 7.1 was briefly introduced at the end of chapter 2. It shows a logical and physical hierarchy of manufacturing capabilities in the shop floor. Capabilities in resource elements reside on machines that are placed in fractal cells on the shop floor. Scheduling occurs at the lowest level of the hierarchy to match the requirement of products with available processing capacities in the system. This integrated framework serves as reference architecture that is applicable to most manufacturing systems. It was developed to capture the features of new manufacturing paradigms, but the architecture remains universally valid to describe more traditional methodologies. For instance a design without hidden machine capabilities would directly map resource elements to machines. A shop floor layout without fractal organisation could be considered as a system with only one fractal cell without a distinction between a fractal and the shop floor. Finally, any advanced scheduling method would primarily aim to match the specific requirements of products to available capacities in the system.

To increase robustness, flexibility and responsiveness to internal and external disturbances multiple new manufacturing methodologies can be deployed simultaneously. Figure 7.1 suggests how integration can be achieved on an abstract level. The reference architecture captures the relationship of the features proposed in the emerging methodologies with respect to the overall system. It supports the claim that amalgamation of the various methodologies is feasible once the described relationships are understood.

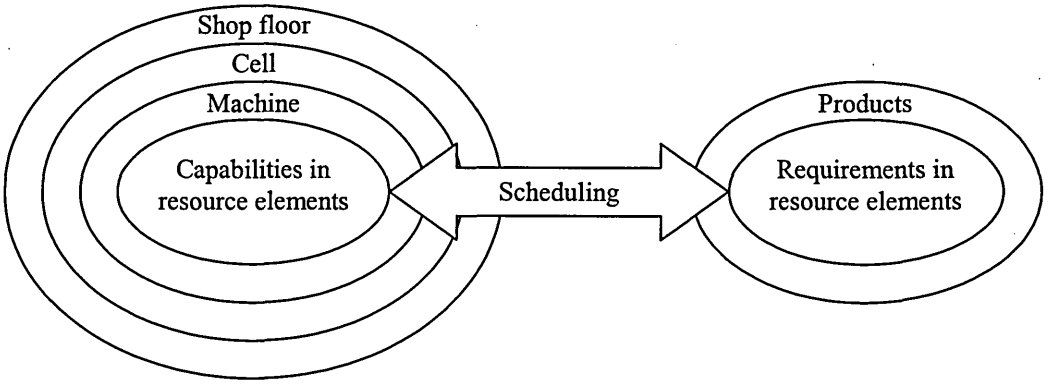



**Figure 7.1** Generalised integration framework for manufacturing concepts

## 7.2 Integration of responsive, fractal and biological concepts

A hypothetical model is developed to illustrate how the integration of fractal, biological and responsive methodologies can be achieved and implemented in practice on a lower level. To integrate cellular structured layouts or fractal cells into biological scheduling, the system entities of dispatcher and exit point are joined to be member of every cell simultaneously, see figure 7.2. Logical membership of a fractal cell is designated as important parameter to attraction fields, and the entry and exit points to the shop floor must belong to every cell for scheduling purposes. For the sake of simplification the dispatcher can be understood as a machine out-queue that is located in the boundaries of every cell. Similarly the exit point where completed products are queued to leave the shop floor can be regarded as a machine in-queue that is part of every cell. Using this simplification, attraction fields can be defined between the following parties, i) an empty transporter and the out-queue of a machine and ii) a product-carrying transporter and the in-queue of a machine.

An empty transporter will prioritise product collection using an objective function based on i) product due dates and ii) the cell associated with the machine out-queue where the product is held relative to its own position. A transporter would be strongly attracted to waiting products in the same cell where it currently resides in order to emphasise intra-cell routings. A product-carrying transporter will also be strongly attracted to resources in the current cell and would only be attracted to in-queues at other cells as last resort. Product-carrying transporters would also include metrics to avoid unloading to busy input-queues. Attraction fields could only be established with a machine in-queue if the

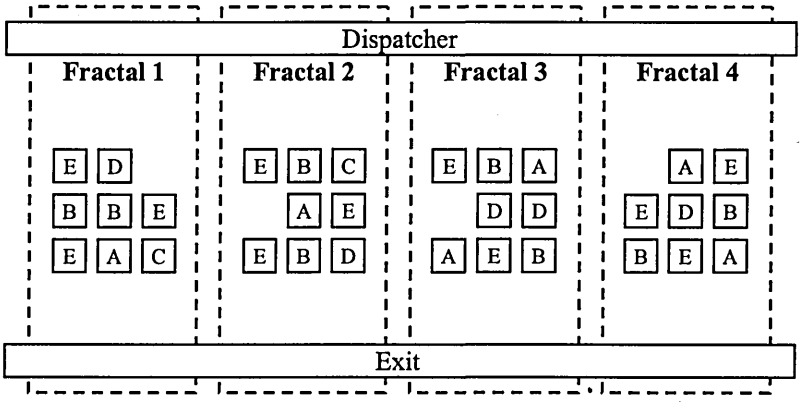



**Figure 7.2** Fractal cells for biological scheduling

machine offers the resource elements sought by the carried product. To avoid unnecessary setup time on machines, a final metric is determined by the current machine setup, or the processing setup needed to complete the last product in the in-queue if this is occupied. Machine in-queues could optionally be modelled to attract transporters that carry products with a more urgent due date.

Figure 7.3 illustrates the resulting relationships by means of attraction fields. Dynamic biological scheduling builds on this model to enhance system responsiveness and to avoid the complexity and rigidity associated with centralised control. Internal disturbances such as machine break downs are reacted to immediately without the need for complex re-scheduling. This type of scheduling introduces the element of competition i) between machines to attract transporters and ii) between transporters over machine capacities. To resolve these contention issues in favour of one particular actor over another, attraction fields are formed with different strengths, represented by a numerical value. This value is based on a composite metric governed by the parameters outlined in figure 7.3. These parameters can be weighted according to business requirements. For instance the importance of cell autonomy or the focus on machine types to avoid set up delays could be adjusted to find a solution that is appropriate for any set of circumstances. Products take no part in scheduling decisions. Instead,

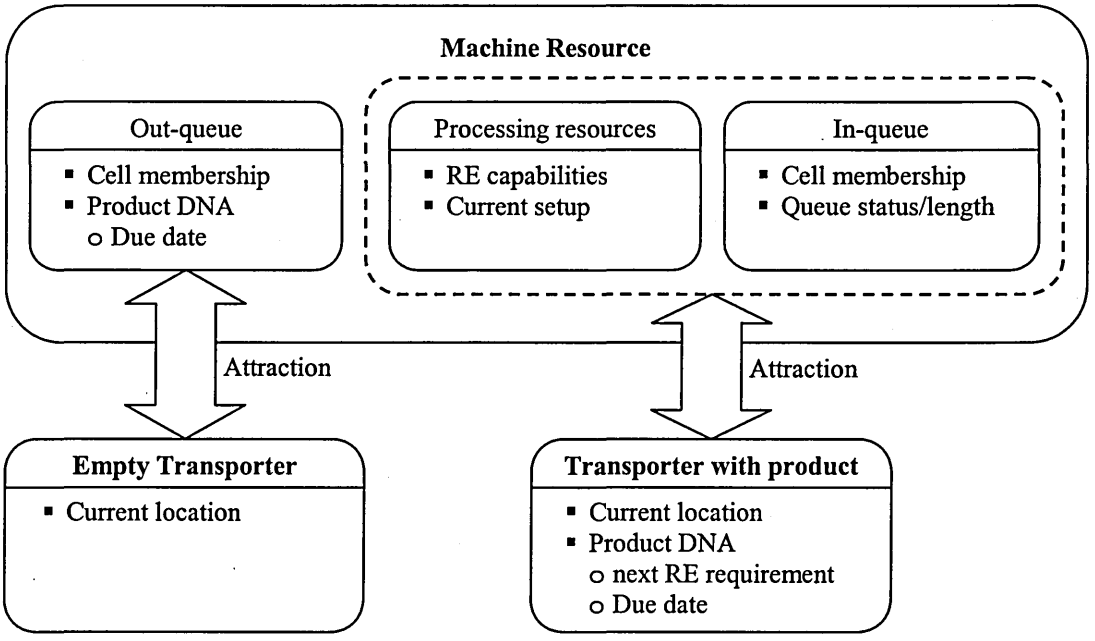



**Figure 7.3** Attraction fields in fractal layouts with resource elements

transporters and machines use the notion of product DNA, which carries instructions on processing in sequential order and due dates.

## 7.3 Simulation and discussion

This integrated model with resource elements and biological scheduling on an optimised layout that was based on a fractal cell organisation was simulated in the Arena software environment. Figure 7.4 shows a system where all machines offer the capabilities of two different resource elements. The speed of transport and handling of material is positively correlated against i) the number of AGVs and ii) the speed of these AGVs in relation to the processing time at machines. The speed and number of AGVs can be crucial to the system throughput time if the level of processing capacity is constant, as shown in figure 7.5. As described in chapter 6, the lead time is a function of the speed of the AGVs, but does not decrease with the number AGVs in the system as demonstrated in figure 7.6 as resource bottlenecks at machines are accentuated. The ideal number of AGVs in the system balances capacities between transport agents and

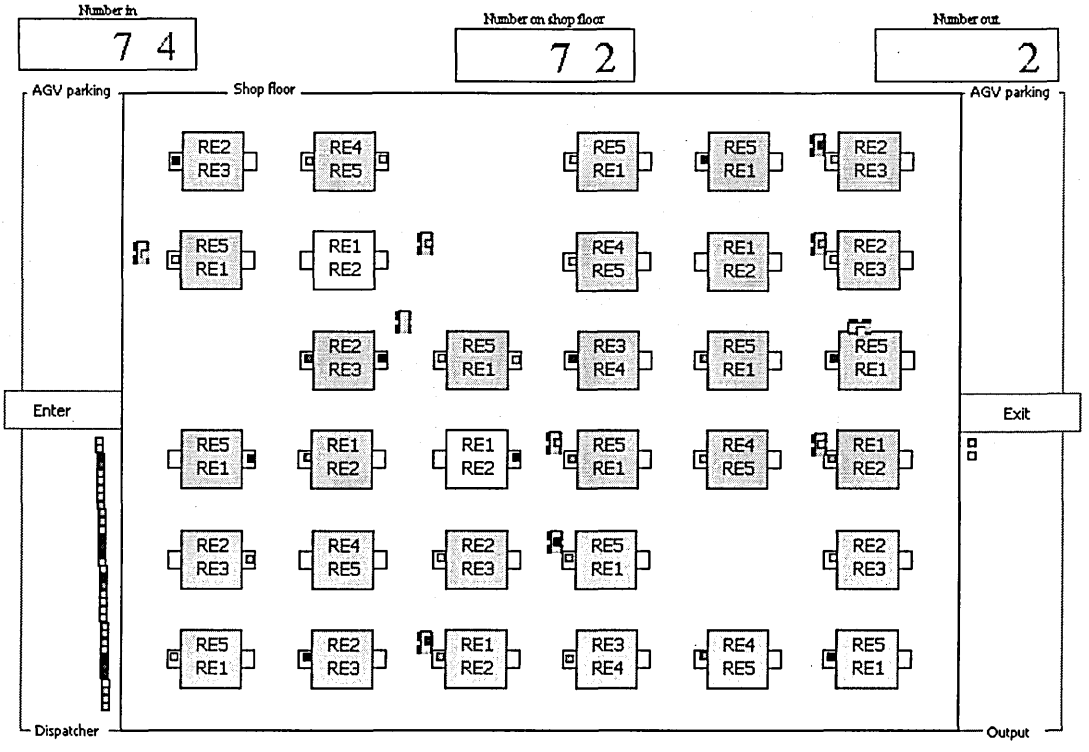



**Figure 7.4** Shop floor layout in the integrated system

resource elements and can be expressed as a function comprising machine quantities, average processing time, AGV speed and average handling distance.

To test the responsiveness of the developed integrated system model changes to the product mix were applied to the simulation. The performance differences in terms of throughput were examined between a biological system without the extended capabilities of resource elements and the proposed system that builds on all three manufacturing methodologies. The results of the simulation runs are illustrated in figures 7.7. The intensity of changes to product mix is plotted in terms of sample

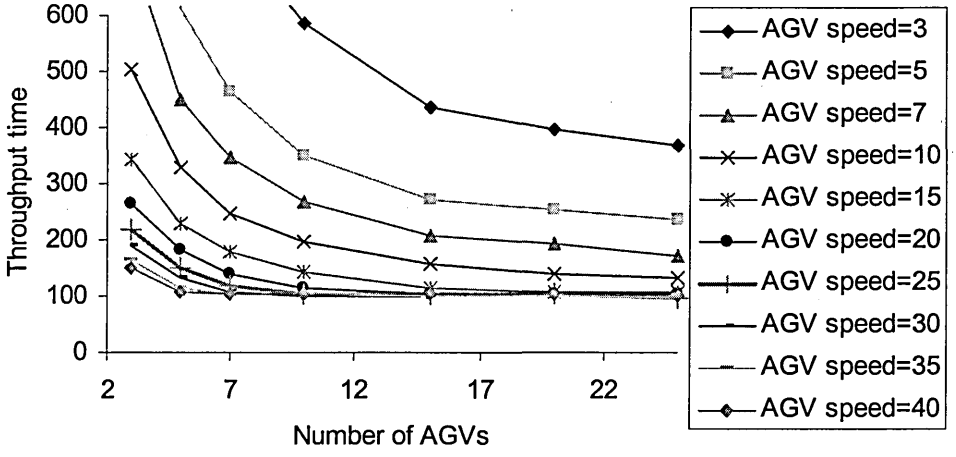



**Figure 7.5** Throughput time in the integrated system model

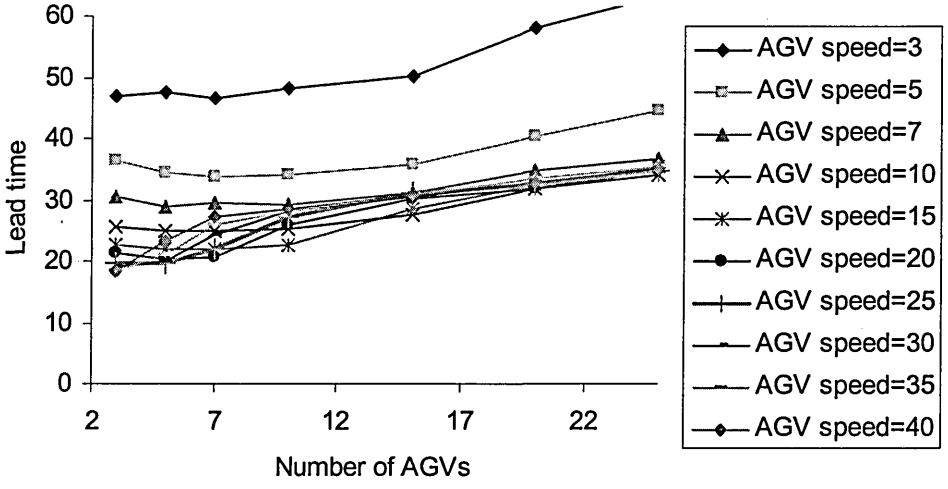



**Figure 7.6** Lead time in the integrated system model

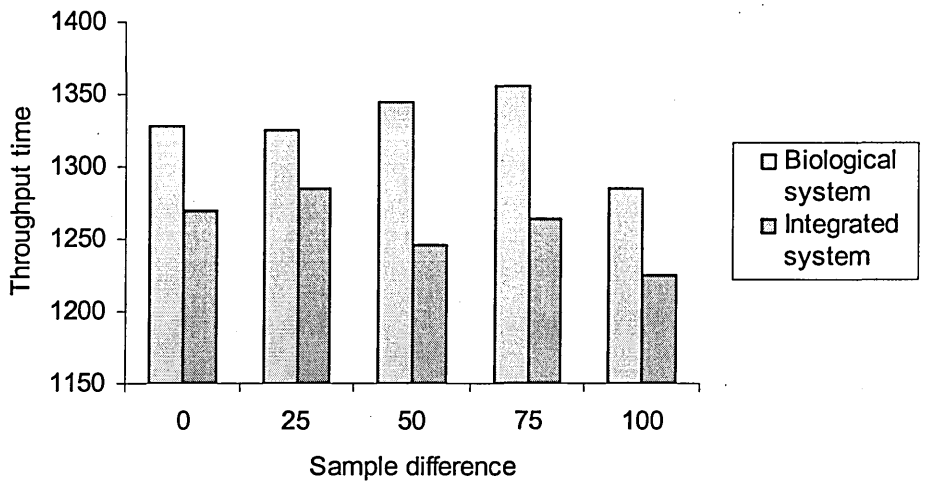



**Figure 7.7** Throughput time under changed product mix

difference that in this instance relates to a difference in the nominal value in product demand for each type in a batch of 2000 total products.

A four to eight per cent improvement to system throughput time can be observed when biological scheduling is permitted to consider hidden resource capabilities. Biological scheduling displays a high degree of flexibility even to the most significant disturbances in product demand levels, whereas the integrated system follows the same trend, but offers a further marginal enhancement in throughput time. The differences between a biological methodology with and without resource elements are expected to be greater when internal disturbances, i.e. machine breakdowns, are considered.

## 7.4 Conclusions

The successful integration of three emerging manufacturing concepts within the same architecture supports the validity of the reference architecture presented in this chapter. The interaction of the inherent features of these concepts corresponds closely to the higher level view provided in the reference architecture. Material handling was improved in multiple ways, i) through an optimised fractal layout of machines, ii) with decentralised self-organisation scheduling that is responsive to changes, iii) with the introduction of attraction fields to replace advanced control and negotiation, and iv)

with an approach centred around resource-elements that takes advantage of hidden machine capabilities to reduce the effect of resource bottlenecks.

The experiments primarily demonstrate that the integration of fractal, biological and responsive manufacturing methodologies is feasible in the theoretical space of computer simulation.

# Chapter Eight - Conclusions

This chapter provides the concluding discussions of the research. It includes a review of the relevant literature and conducted research. It also comprises a discussion of research findings and their broader implications. Later in the chapter, the limitations of the research methods and findings are discussed. Next, the contributions of the research to knowledge in the field of the 21st century manufacturing systems are summarised. The chapter ends with some suggestions for the future work.

## 8.1 Review of conducted research

Since the 1990's the rapidly changing business environment has forced manufacturing organisations to look for radical new methods to improve the performance and cost-effectiveness of their processes. As a result of this growing interest many new organisational concepts especially designed to meet the requirements of an unstable and unpredictable marketplace have emerged. Among the most promising and well-known new concepts are agile, holonic, fractal, biological and responsive manufacturing systems. Although the potential of these new theories have been widely recognised in the academic literature only a handful of research papers have been published in the area. In view of that, this research aimed to improve the comprehension and implementation of these 21st century manufacturing systems by conducting a comprehensive comparison of the concepts and developing an integrated reference architecture by merging the distinctive and most advantageous features of the paradigms within the scope of manufacturing shop floor operations. Based on the collected information about the strength and weaknesses of each concept, fractal configuration, biological scheduling and the responsive manufacturing concept of presenting machine capabilities and product processing requirements as resource elements were selected as the key concepts of the new 21st century manufacturing system. To enable the integration, each concept required further development regarding the specific functions assigned to them.

Since its introduction by Warnecke (1993), the research efforts into the concept of fractal manufacturing have focused on layout design using "fractal units" as self-similar and dynamic building blocks of the system. Leading research in this area has been conducted by Venkatadri *et al.* (1997), who proposed a methodology for fractal cell design. However, the authors failed to clearly define the multiple design parameters involved in the fractal cell configuration. This significantly restricted the application of their methodology. In this thesis, various fractal cell configuration methods for different system design objectives and constraints were proposed. Number of parameters that determine the level of interaction between the cells, the distribution of different product types among the cells and the similarity of cell capabilities were discussed and simulated. Layout optimisation through the used meta-heuristic procedures yielded a noticeable reduction in handling distances. The experiments indicated that arranging the

position of resources to meet the requirements for a specific product demand and mix can significantly reduce material travelling distances even if the demand for the different products is unprdictable. Fractal layouts also provided simpler material flows and system control.

The biological manufacturing concept, as proposed by Ueda (1992), aims to imitate the adaptable structures and behaviour of biological organisms in manufacturing systems. The self-organisation simulator suggested by Ueda *et al.* (1997b) and Vaario and Ueda (1998a) was designed to direct configuration, reconfiguration and scheduling activities in real-time by continuously calculating the local potential fields of the machines and transporters on the shop floor. However, the continuous competition between system elements resulted in a chaotic behaviour. This research proposed a new approach to biological manufacturing based on wireless communication between system components to facilitate better control. The methodology enabled a flexible and responsive reaction to any disturbances on the shop floor. The proposed strategies for product routing and deadlock-avoidance proved to be feasible under simulation condition and did respond extremely well to disturbances.

The responsive manufacturing concept (Gindy and Saad 1998) includes a framework for representing manufacturing resources and their capabilities using resource elements in process planning and scheduling. In this thesis, new rules were proposed in order to integrate the resource elements with the proposed biological self-organised secheduling and fractal layouts. The conducted experiments confirm that resource elements are applicable to dynamic scheduling systems. Internal disturbances through machine breakdown events were balanced out efficiently. Hence, it was established that resource elements can improve system flexibility and responsiveness in the conditions of dynamic self-organisation and are suitable for 21st century manufacturing systems.

These new concepts reshape our understanding of the manufacturing process. Each methodology offers benefits in key areas, but limitations in their application are recognised. A combined model incorporating key aspects and advantages of all three concepts, biological, fractal and responsive manufacturing was put forward and the resulting system was simulated.

## 8.2 Discussion

The principle objective of this work was to research the emerging manufacturing systems, namely fractal, biological and responsive manufacturing, and to develop an integrated reference architecture that achieves a combination of their distinctive features. This would enable manufacturing enterprises to successfully handle the configuration/reconfiguration, planning and control activities under conditions of uncertainty and continuous change. The new system would allow manufacturing organisations to adapt and to find a rapid and balanced response to changing customer requirements.

In the course of the research, a detailed investigation of fractal, biological and responsive manufacturing systems was conducted. It provided a clear account of the strengths and weaknesses of each concept. A comparison of these new approaches identified a number of common and distinct features that would enable the integration of the concepts and the creation of an adaptive reference architecture for the 21st century manufacturing systems. The concept of fractal configuration, biological scheduling and the 'resource elements' representation of resource capabilities and product processing requirements were selected as the major elements of the new system. A detailed study of fractal layout design resulted in seven distinctive methods for structuring and operating fractal cellular systems. A methodology was developed for scheduling and control in biological manufacturing systems. Three options were presented to achieve varying levels of responsiveness to disturbances and complexity of control. Resource elements were used in both fractal layouts and biological systems to optimise the resource utilisation and to improve the system performance. The integrated system was based on an optimised shop floor layout taken from the study of fractal cells and used dynamic self-organisation scheduling to match the processing requirements and capabilities using resource elements.

Experimentation with computer simulation representative of manufacturing systems indicated an improvement in system throughput each time one of the features of the new manufacturing methodologies was added on top of the architecture. The resulting architecture responded extremely well to disturbances in the system.

## 8.3 Limitations

The developed methodologies and the integrated framework were implemented and experimented using one case study. Although the hypothetical case study was selected to reflect the requirements assumed for emerging manufacturing organisations with multiple product types and varying order quantities, the results of this study should be viewed with this limitation in mind. In addition, several assumptions were made during the implementation and experimentation of the methodologies, which have been discussed in the relevant sections on the thesis. Moreover, the implementation of the proposed procedures and the reference model in a real factory system would require significant investments in terms of technology and training on work practices. Hence, it can be recognised that the primary achievement of this work is the provision of ideas for future development in the academic space.

## 8.4 Research contributions

The research has established a reference architecture that merges the features and benefits of fractal layouts with biological scheduling that considers the responsive manufacturing concept of resource elements. The research results indicate that the performance of the biological scheduling system can be improved by using hidden capabilities of resources as represented by resource elements. The procedure for optimising resource arrangements in fractal layouts reduced material travelling distances. The integration of the resource elements concept enabled a better distribution of processing capabilities in the system.

The key achievements of the research are:

a) An analysis of the strength and weaknesses of the next generation of manufacturing systems
b) A comprehensive comparison of the design and operational and organisational features of fractal, holonic, biological and responsive manufacturing concepts
c) A methodology for designing fractal layouts with different organisational objectives
d) A mathematical model and the implementation of the seven fractal cell configuration methods in C++ using tabu-search
e) A flexibility study of fractal layouts

f) The development and implementation of a distributed real-time scheduling and control methodology for biological manufacturing systems with deadlock avoidance using Arena simulation software
g) A procedure for representing resource capabilities and product processing requirements as resource elements in biological systems
h) A procedure for matching capabilities and requirements in biological manufacturing systems using resource elements
i) The proposal of an integrated reference architecture that incorporates fractal, biological and responsive manufacturing concepts
j) A working software model that simulates the implementation of this reference architecture

## 8.5 Future work

In the course of the research project it became apparent that only a limited amount of research into the emerging manufacturing systems had previously been conducted. Thus, many of the new approaches are still fairly abstract concepts. This research has considered only three of the new paradigms within a limited scope of manufacturing shop floor layout design and scheduling. However, it was still possible to identify several areas for future research within the scope of this research. Some of the most significant topics are discussed next.

a) Work on the flexibility of fractal layouts to react to changes in product types needs to be continued.
b) The responsiveness of fractal layouts to random machine breakdowns or changes to the overall order volume can be evaluated.
c) Fractal machine layout on the shop floor could consider hidden capabilities, but this is expected to reduce the importance of layout optimisation.
d) A comparative study with simulation between fractal layouts and other layouts can be conducted to more precisely determine the merit of arbitrary cell boundaries in terms of controllability and flexibility to disturbances.
e) Resource utilisation could be further improved through the development of control techniques for sequencing of products at the input dispatcher when biological scheduling is used. However, this approach requires knowledge of the

overall system state and in principle conflicts with some of the design goals of this methodology.

f) A formula to compute the optimal number of AGVs in biological scheduling can be created, but the development could be complex due to the number of variables involved.

g) The performance of the identified different types of dynamic biological scheduling approaches need to be evaluated and compared.

h) A methodology for preventing system blockage ever occurring in biological systems need to be developed. This could be achieved by controlling the number of products in the system or having an emergency AGV.

i) To include setup delays to resource element based scheduling and to determine the performance difference between a system with resource elements and setup delays and a system with low resource utilisation due to hidden underutilised capabilities.

j) Development of a methodology to minimise resource element setups in dynamic biological scheduling.

k) To investigate if any benefits can be obtained by considering the hidden resource capabilities (presented as resource elements) during the fractal layout design.

l) To study the impact of machine layout on system performance when biological scheduling is used.

m) To conduct more experiments for different case studies to add further weight to the conclusions that were reached.

# References

Agarwal, A. and Sarkis, J., 1998, A review and analysis of comparative performance studies on functional and cellular manufacturing layouts. *Computers & Industrial Engineering*, 34, 1, 77-89.

Akright, W. T. and Kroll, D. E., 1998, Cell formation performance measures – Determining when to change an existing layout. *Computers & Industrial Engineering*, 34, 1, 159-171.

Akturk, M. S. and Turkcan, A., 2000, Cellular manufacturing system design using a holonistic approach. *International Journal of Production Research*, 38, 10, 2327-2347.

Alvarez, E. and Diaz, F., 2004, An application of a real-time scheduling system for turbulent manufacturing environments. *Robotics and Computer-Integrated Manufacturing,* 20, 485-494.

Arai, T., Aiyama, Y., Sugi, M. and Ota, J., 2001, Holonic assembly system with plug and produce. *Computers in Industry*, 46, 289-299.

Arteta, B. M. and Giachetti, R. E., 2004, A measure of agility as the complexity of the enterprise system. *Robotics and Computer Integrated Manufacturing*, 20, 495-503.

Askin, R. G., Ciarallo, F. W. and Lundgren, N. H., 1999, An empirical evaluation of holonic and fractal layouts. *International Journal of Production Research*, 37, 5, 961-978.

Askin, R. G., Lundgren, N. H. and Ciarallo, F., 1996, A material flow based evaluation of layout alternatives for agile manufacturing. In R. J. Graves, L. F. McGinnis, D. J. Medeiros, R. E. Ward and M. R. Wilhelm (eds.), *Progress in material handling research* (Braun-Brumfield), pp. 71-90.

Azadivar, F. and Wang, J., 2000, Facility layout optimization using simulation and genetic algorithms. *International Journal of Production Research*, 38, 17, 4369-4383.

Babiceanu, R. F., Chen, F. F. and Sturges, R. H., 2005, Real-time holonic scheduling of materials handling operations in a dynamic manufacturing environment. *Robotics and Computer-Integrated Manufacturing,* 21, 328-337.

Balakrishnan, J. and Cheng, C. H., 1998, Dynamic layout algorithms: a state-of-the-art survey. *Omega – The International Journal of Management Science,* 26, 4, 507-521.

Balasubramanian, S., Brennan, R. W. and Norrie, D. H., 2001, An architecture for metamorphic control of holonic manufacturing systems. *Computers in Industry*, 46, 13-31.

Balasubramanian, S., Zhang, X. and Norrie, D. H., 2000, Intelligent control for holonic manufacturing systems. *Proceedings of the Institute of Mechanical Engineers - Part B - Journal of Engineering Manufacture*, 214, 953-961.

Bard, J. F., Dar-El, E. and Shtub, A., 1992, An analytic framework for sequencing mixed model assembly lines. *International Journal of Production Research*, 30, 1, 35-48.

Bartezzaghi, E., 1999, The evolution of production models: is a new paradigm emerging? *International Journal of Operations & Production Management*, 19, 2, 229-250.

Bastos, R. M., de Oliveira, F. M. and de Oliveira, J. P. M., 2005, Autonomic computing approach for resource allocation. *Expert Systems with Applications,* 28, 9-19.

Baykasoglu, A., 2003, Capability-based distributed layout approach for virtual manufacturing cells. *International Journal of Production Research*, 41, 11, 2597-2618.

Baykasoglu, A. and Gindy, N. N. Z., 2000, MOCACEF 1.0: multiple objective capability based approach to form part-machine groups for cellular manufacturing applications. *International Journal of Production Research*, 38, 5, 1133-1161.

Benjaafar, S. and Sheikhzadeh, M., 2000, Design of flexible plant layouts. *IIE Transactions*, 32, 4, 309-322.

Benjaafar, S., Heragu, S. S. and Irani, S. A., 2002, Next generation factory layouts: research challenges and recent progress. *Interfaces,* 32, 6, 58-76.

Berggren, C., 1994, Nummi vs. Uddevalla. *Sloan Management Review*, Winter issue, 37-45.

Billo, R. E., 1998, A design methodology for configuration of manufacturing cells. *Computers & Industrial Engineering*, 34, 1, 63-75.

Bongaerts, L., Jordan, P., Timmermans, P., Valckenaers, P. and Wyns, J., 1997a, Evolutionary development in shop floor control. *Computers in Industry*, 33, 295-304.

Bongaerts, L., Monostori, L., McFarlane, D. and Kadar, B., 2000, Hierarchy in distributed shop floor control. *Computers in Industry*, 43, 123-137.

Bongaerts, L., van Brussel, H., Valckenaers, P. and Peeters, P., 1997b, Reactive scheduling in holonic manufacturing systems: architecture, dynamic model and co-operation strategy. *Proceedings of advanced summer institute of the network of excellence on intelligent control and integrated manufacturing ASI97*, Budapest, Hungary, 14-17 July.

Bonney, M., 2000, Reflections on production planning and control (PPC). *Gestao & Producao*, 7, 3, 181-207.

Brehmer, N. and Martinetz, J., 2001, Conceptual model of information navigation in networked organisations. (available at: http://www.adrenalin-company.de/images/pdf/adrenalin_e2001_paper.pdf) (accessed on July 2002).

Brennan, R. W., 2000, Performance comparison and analysis of reactive and planning-based control architectures for manufacturing. *Robotics and Computer Integrated Manufacturing,* 16, 191-200.

Brennan, R. W. and Norrie, D. H., 2001, Evaluating the performance of reactive control architecture for manufacturing production control. *Computers in Industry,* 46, 235-245.

Brezocnik, M. and Balic, J., 2001, A genetic-based approach to simulation of self-organizing assembly. *Robotics and Computer Integrated Manufacturing*, 17, 113-120.

Buchel, A., Breuil, D. and Doumeingts, G., 1984, Comparison of design methodologies, characteristics and deficiencies. *Production Management Systems: Strategies and Tools for Design.*

Bukchin, J., 1998, A comparative study of performance measures for throughput of a mixed model assembly line in a JIT environment. *International Journal of Production Research*, 36, 10, 2669-2685.

Bullinger, H. J., Fremerey, F. and Fuhrberg-Baumann, J., 1995, Innovative production structures - precondition for a customer-oriented production management. *International Journal of Production Economics*, 41, 15-22.

Burkel, J., 1991, Applying CIM for competitive advantage. *Proceedings of autofact'91,* Chicago, USA.

Bussmann, S., 1998, An agent-oriented architecture for holonic manufacturing control. *Proceedings of IMS'98-ESPRIT workshop in intelligent manufacturing systems*, Lausanne, Switzerland.

Chalmeta, R., Campos, C. and Grangel, R., 2001, References architectures for enterprise integration. *The Journal of Systems and Software,* 57, 175-191.

Chase, R. B. and Aquilano, N. J., 1992, *Production & operations management - A life cycle approach,* 6th Ed. (Irwin: Boston, Massachusetts, USA).

Choi, B. K. and Kim, B. H., 2000, Human-centred virtual manufacturing systems for next generation manufacturing. *Proceedings of 2000 international CIRP design seminar*, Haifa, Israel, May 16-18, 169-174.

Christian, I., Ismail, H., Mooney, J., Snowden, S., Toward, M. and Zhang, D., 2001, Agile manufacturing transitional strategies. *Proceedings of the 4th SMESME international conference*, Aalborg, Denmark, 14-16 May.

Clark, K. B. and Fujimoto, T., 1991, *Product development performance: strategy, organization and management in the world auto industry* (Harvard University Publications, Cambridge, Massachusetts, USA).

Co, H. C. and Araar, A., 1988, Configuring cellular manufacturing systems. *International Journal of Production Research,* 26, 9, 1511-1522.

Conboy, K. and Fitzgerald, B., 2004, Towards a conceptual framework of agile methods: A study of agility in different disciplines. *Proceedings of the WISER'04 conference*, Newport Beach, California, USA, 5 November, pp. 37-44.

Csaszar, P., Tirpak, T. M. and Nelson, P. C., 2000, Optimization of a high-speed placement machine using tabu search algorithms. *Annals of Operations Research,* 96, 125-147.

Cusumano, M. A., 1994, The limits of "lean". *Sloan Management Review*, Summer issue, 27-32.

Davis, E., 1995, What's on American managers' minds? *Management Review,* 4, 14-20.

Debnar, R., Kosturiak, J. and Matuszek, J., 2001, A methodology for design of fractal company. (available at: http://www. iizp.uz. zgora. pl/ zlsi / data / mim / debnar. zip) (accessed on April 2003).

Demeester, L., Eichler, K. and Loch, C. H., 2004, Organic production systems: what the biological cell can teach us about manufacturing. *Manufacturing & Service Operations Management*, 6, 2, 115-132.

DeVor, R., Graves, R. and Mills, J. J., 1997, Agile manufacturing research: accomplishments and opportunities. *IIE Transactions*, 29, 813-823.

Doll, W. J. and Vonderembse, M. A., 1992, The evolution of manufacturing systems: towards the post-industrial enterprise. In *Manufacturing strategy - process and content*, Voss, C. A. (ed.) (London: Chapman & Hall), pp. 353-370.

D'Souza, D. E. and Williams, F. P., 2000, Towards a taxonomy of manufacturing flexibility dimensions. *Journal of Operations Management,* 18, 577-593.

Duguay, C. R., Landry, S. and Pasin, F., 1997, From mass production to flexible/agile production. *International Journal of Operations & Production Management*, 17, 12, 1183-1195.

Duplaga, E. A. and Bragg, D. J., 1998, Mixed-model assembly line sequencing heuristics for smoothing component parts usage: a comparative analysis. *International Journal of Production Research*, 36, 8, 2209-2224.

Ellegard, K., Jonsson, D., Engstrom, T., Johansson, M. I., Medbo, L. and Johansson, B., 1992, Reflective production in the final assembly of motor vehicles - an emerging Swedish challenge. *International Journal of Operations & Production Management*, 12, 7/8, 117-133.

England, D., 2004, *Operational planning of discrete component manufacturing lines.* PhD thesis. Loughborough University, UK.

Esmail, K. K. and Saggu, J., 1996, A changing paradigm. *Manufacturing Engineering,* 75, 6, 285-288.

Farrington, P. A. and Nazemetz, J. W., 1998, Evaluation of the performance domain of cellular and functional layouts. *Computers & Industrial Engineering*, 34, 1, 91-101.

Featherston, S., 1999, Study of reasons for the adoption of lean production in the automobile industry: questions for the AEC industries. *Proceedings of the 7th annual conference of the international group for lean construction (IGLC-7)*, Berkeley, California, USA, 26-28 July.

Fischer, K., 1999, Agent-based design of holonic manufacturing systems. *Robotics and Autonomous Systems*, 27, 3-13.

Fujii, N., Hatono, I. and Ueda, K., 2004, Reinforcement learning approach to self-organization in a biological manufacturing system framework. *Proceedings of the Institute of Mechanical Engineers - Part B - Journal of Engineering Manufacture*, 218, 6, 667-673.

Fujii, N., Vaario, J. and Ueda, K., 1997, Potential field based simulation of self-organization in biological manufacturing systems. *Proceedings of manufacturing system design '97*, Magdeburg, Germany, 14-16 May.

Gau, K. Y. and Meller, R. D., 1999, An iterative facility layout algorithm. *International Journal of Production Research*, 37, 16, 3739-3758.

Giebels, M., Kals, H. and Zijm, H., 1999, Building holarchies for concurrent manufacturing planning and control. *Proceedings of the 2nd international workshop on intelligent manufacturing systems*, Leuven, Belgium, 22-24 September.

Gindy, N. N. Z. and Ratchev, T. M., 1997, Cellular decomposition of manufacturing facilities using resource elements. *Integrated Manufacturing Systems,* 8, 4, 215-222.

Gindy, N. N., Ratchev, T. M. and Case, K., 1996, Component grouping for cell formation using resource elements. *International Journal of Production Research*, 34, 3, 727-752.

Gindy, N. N. and Saad, S. M., 1998, Flexibility and responsiveness of machining environments. *Integrated Manufacturing Systems*, 9, 4, 218-227.

Gindy, N. N., Saad, S. M. and Yue, Y., 1999, Manufacturing responsiveness through integrated process planning and scheduling. *International Journal of Production Research*, 37, 11, 2399-2418.

Giordano, F. R., Weir, M. D. and Fox, W. P., 2003, *First course in mathematical modeling*, 3rd Ed. (Brooks Cole).

Glover, F., 1989, Tabu search Part I. *ORSA Journal on Computing,* 1, 190-206

Glover, F., 1990, Tabu search Part II. *ORSA Journal on Computing,* 2, 4-31

Goldman, S., Nagel, R. and Preiss, K., 1995, *Agile competitors and virtual organizations* (Van Nostrand Reinhold: New York).

Gould, P., 1997, What is agility? *Manufacturing Engineer*, 76, 1, 28-31.

Gou, L., Luh, P. B. and Kyoya, Y., 1998, Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation. *Computers in Industry*, 37, 213-231.

Griffiths, J., James, R. and Kempson, J., 2000, Focusing customer demand through manufacturing supply chains by the use of customer focused cells: An appraisal. *International Journal of Production Economics*, 65, 111-120.

Gunasekaran, A., 1998 Agile manufacturing: enablers and an implementation framework. *International Journal of Production Research*, 36, 5, 1223-1247.

He, D., Babayan, A. and Kusiak, A., 2001, Scheduling manufacturing systems in an agile environment. *Robotics and Computer Integrated Manufacturing*, 17, 87-97.

Hines, P., Holweg, M. and Rich, N., 2004, Learning to evolve - A review of contemporary lean thinking. *International Journal of Operations & Production Management*, 24, 10, 994-1011.

Hitt, M. A., 2000, Transformation of management for the new millennium. *Organizational Dynamics,* 28, 3, 7-16.

Höpf, M.,1994, Holonic manufacturing systems (HMS) - The basic concepts and a report of IMS test case 5. (available at: http://hms.ifw.uni-hannover.de/public/Feasibil/holo2.htm) (accessed on August 2003).

Hopp, W. J. and Spearman, M. L., 2000, *Factory physics*. 2nd Ed. (New York: McGraw-Hill).

Hughes, J. J., 1997, Views of the future. In Wolkoff, R. L. (ed.), Next-generation manufacturing: A framework for action. The Agility forum, Bethlehem, PA, USA.

Inman, R. R. and Schmeling, D. M., 2003, Algorithm for agile assembling-to-order in the automotive industry. *International Journal of Production Research*, 41, 16, 3831-3848.

Irani, S. A., Cavalier, T. M. and Cohen, P. H., 1993, Virtual manufacturing cells: exploiting layout design and intercell flows for the machine sharing problem. *International Journal of Production Research*, 31, 4, 791-810.

Iwata, K., Onosato, M. and Koike, M., 1994, Random manufacturing system: a new concept of manufacturing systems for production to order. *Annals of the CIRP*, 43, 1, 379-383.

Jackson, M. and Johansson, C., 2003, An agility analysis from a production system perspective. *Integrated Manufacturing Systems*, 14, 6, 482-488.

Jin-Hai, L., Anderson, A. R. and Harrison, R. T., 2003, The evolution of agile manufacturing. *Business Process Management Journal*, 9, 2, 170-189.

Kadar, B., Monostori, L. and Szelke, E., 1998, An object-oriented framework for developing distributed manufacturing architectures. *Journal of Intelligent Manufacturing*, 9, 173-179.

Kalpakjian, S. and Schmid, S. R., 2001, *Manufacturing engineering and technology*. 4th Ed. (New Jersey: Prentice-Hall).

Kannan, V. R. and Ghosh, S., 1996, Cellular manufacturing using virtual cells. *International Journl of Operations & Production Management*, 16, 5, 99-112.

Katalinic, B. and Kordic, V., 2002, Bionic assembly systems: Autonomous agents in self-organising complex flexible assembly system in CIM environment. *Proceedings of the 4th international workshop on emergent synthesis (IWES'02)*, Kobe, Japan, 9-10 May.

Katalinic, B., Visekruna, V. and Kordic, V., 2002, Bionic assembly systems: design and scheduling of next generation of self-organising complex flexible assembly system in CIM environment. *Proceedings of the 35th CIRP international seminar on manufacturing systems*, Seoul, Korea, 12-15 May.

Katayama, H. and Bennett, D., 1996, Lean production in a changing competitive world: a Japanese perspective. *International Journal of Operations & Production Management*, 16, 2, 8-23.

Katayama, H. and Bennett, D., 1999, Agility, adaptability and leanness: a comparison of concepts and a study of practices. *International Journal of Production Economics*, 60/61, 43-51.

Kidd, P. T., 2000, Next generation manufacturing enterprise model. (available at: http://www.cheshirehenbury.com/agility/ngemodel.html) (accessed on September 2005).

Kimberley, W., 2001, Skoda: An eastern European success. *Automotive Manufacturing & Production*, 113, 6, 26-28.

Klopp, M., Kissling, W. and Spiewack, M., 1997, Fractal lines. *Manufacturing Engineer*, 76, 2, 82-85.

Ko, K. C. and Egbelu, P. J., 2003, Virtual cell formation. *International Journal of Production Research,* 41, 11, 2365-2389.

Kochikar, V. P. and Narendran, T. T., 1998, Logical cell formation in FMS, using flexibility-based criteria. *The International Journal of Flexible Manufacturing Systems,* 10, 163-182.

Koestler, A., 1967, *The ghost in the machine* (London: Hutchinson).

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G. and Van Brussel, H., 1999, Reconfigurable manufacturing systems. *Annals of CIRP,* 48, 2, 527-540.

Koren, Y., Hu, S. J. and Weber, T. W., 1998, Impact of manufacturing system configuration on performance. *Annals of the CIRP*, 47, 1, 369-372.

Koste, L. L. and Malhotra, M. K., 1999, A theoretical framework for analyzing the dimensions of manufacturing flexibility. *Journal of Operations Management,* 18, 75-93.

Koylu, R., 2000, Tabu search. *Proceedings of IE572 production planning systems design conference.* Spring 2000.

Krajewski, L. J. and Ritzman, L. P., 2005, *Operations management: Processes and value chains.* (New Jersey: Pearson Prentice Hall).

Kuehnle, H., 1995, Guidelines for future manufacturing - Necessity of a change of organizational structures in industry and ways to the "fractal company". *Web Journal.* (available at: http://fstroj.utc.sk/journal/engl/papers.html) (accessed on July 2002).

Lacksonen, T. A., 1997, Preprocessing for static and dynamic facility layout problems. *International Journal of Production Research,* 35, 4, 1095-1106.

Lee, S. D., Huang, K. H. and Chiang, C. P., 2001, Configuring layout in unidirectional loop manufacturing systems. *International Journal of Production Research,* 39, 6, 1183-1201.

Levary, R. R., 1992, Enhancing competitive advantage in fast-changing manufacturing environment. *Industrial Engineering*, 24, 12, 21-28.

Lim, M. K. and Zhang, D. Z., 2004, An integrated agent-based approach for responsive control of manufacturing resources. *Computers & Industrial Engineering,* 46, 221-232.

Little, D. and Rollins, R., 2000, Integrated planning and scheduling in the engineer-to-order sector. *International Journal of Computer Integrated Manufacturing*, 13, 6, 545-554.

Maione, B. and Naso, D., 2001, Evolutionary adaptation of dispatching agents in heterarchical manufacturing systems. *International Journal of Production Research,* 39, 7, 1481-1503.

Manzini, R., Gamberi, M., Regattieri, A. and Persona, A., 2004, Framework for designing a flexible cellular assembly system. *International Journal of Production Research,* 42, 17, 3505-3528.

Markfort, D., Martinetz, J.,Klostermeyer, A. and Brehmer, N., 2000, Advanced fractal companies use information supply chains (ADRENALIN). (available at: http://www.adrenalin-company.de/images/pdf/paper_e2000_madrid.pdf) (accessed on July 2002).

Markus, A., Kis Vancza, T. and Monostori, L., 1996, A market approach to holonic manufacturing. *Annals of the CIRP*, 45, 1, 433-436.

Mathias, P., 1983, *The first industrial nation: An economic history of Britain 1700-1914.* 2nd Ed. (London: Routledge).

McCarthy, I. and Tsinopoulos, C., 2003, Strategies for agility: an evolutionary and configurational approach. *Integrated Manufacturing Systems*, 14, 2, 103-113.

McCormack, R., 2000, Honda reduces manufacturing costs by 87 percent with prototype biological manufacturing system. *Manufacturing News*, 7, 9.

McCurry, L. and McIvor, R., 2002, Agile manufacturing: 21st century strategy for manufacturing on the periphery? *Irish Journal of Management*, 23, 2, 75-93.

McFarlane, D. C., and Bussmann, S., 2000, Developments in holonic production planning and control. *Production Planning and Control*, 11, 6, 522-536.

McLean, C. R., Bloom, H. M. and Hopp, T. H., 1982, The virtual cell. *Proceedings of 4th IFAC/IFIP conference on information control problems in manufacturing technology*, Gaithersburg, MD, October, 207-215.

Mehrabi, M. G., Ulsoy, A. G. and Koren, Y., 2000, Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent Manufacturing,* 11, 403-419.

Meller, R. D., 1997, *Multi-channel manufacturing.* Technical report (Auburn: Department of industrial and systems engineering, Auburn university).

Meng, G., Heragu, S. S. and Zijm, H., 2004, Reconfigurable layout problem. *International Journal of Production Research,* 42, 22, 4709-4729.

Meredith, S. and Francis, D., 2000, Journey towards agility: the agile wheel explored. *The TQM Magazine*, 12, 2, 137-143.

Mertins, K., Friedland, R. and Rabe, M., 2000, Capacity assignment of virtual manufacturing cells by applying lot size harmonization. *International Journal of Production Research*, 38, 17, 4385-4391.

Mill, F. and Sherlock, A., 2000, Biological analogies in manufacturing. *Computers in Industry*, 43, 153-160.

Molleman, E., Slomp, J. and Rolefes, S., 2002, The evolution of a cellular manufacturing system – a longitudinal case study. *International Journal of Production Economics,* 75, 305-322.

Montreuil, B., Venkatadri, U. and Rardin, R. L., 1999, Fractal layout organization for job shop environments. *International Journal of Production Research*, 37, 3, 501-521.

Moskal, B. S., 1995, Son of agility. *Industry Week*, 244, 10, 12-17.

Muffatto, M., 1999, Evolution of production paradigms: the Toyota and Volvo cases. *Integrated Manufacturing Systems*, 10, 1, 15-25.

Nagel, R. N. and Dove, R., 1991, *21st century manufacturing enterprise strategy: an industry-led view of agile manufacturing.* (Iacocca Institute, Lehigh University, Bethlehem, PA, USA).

NGMS-IMS consortium, 2000, *NGMS-IMS project - Interim report* (CAM-I Inc., Bedford, Texas, USA).

Noori, H. and Lee, W. B., 2000, Fractal manufacturing partnership: exploring a new form of strategic alliance between OEMs and suppliers. *Logistics Information Management*, 13, 5, 301-311.

Ohkura, K., Sera, K. and Ueda, K., 1999, A learning multi-agent approach to dynamic scheduling for autonomous distributed manufacturing systems. *Proceedings of 15th international conference on computer-aided production engineering (CAPE99)*, pp. 325-330.

Okino, N., 1992, A prototyping of bionic manufacturing system. *Proceedings of an international conference on object oriented manufacturing systems*, Calgary, Canada, 297-302.

Oliver, N., Delbridge, R., Jones, D. and Lowe, J., 1994, World class manufacturing: further evidence in the lean production debate. *British Journal of Management*, 5, 53-63.

Ozcelik, F. and Islier, A. A., 2003, Novel approach to multi-channel manufacturing system design. *International Journal of Production Research*, 41, 12, 2711-2726.

Pagell, M. and Krause, D. R., 2004, Re-exploring the relationship between flexibility and the external environment. *Journal of Operations Management*, 21, 629-649.

Perry, D. E. and Wolf, A. L., 1992, Foundations for the study of software architecture. *ACM SIGSOFT,* 17, 4, 40-52.

Quinn, R. D., Causey, G. C., Merat, F. L., Sargent, D. M., Barendt, N. A., Newman, W. S., Velasco, V. B., Podgurski, A., Jo, J. Y., Sterling, L. S. and Kim, Y., 1997, An agile manufacturing workcell design. *IIE Transactions*, 29, 901-909.

Rahimifard, S., Newman, S. T. and Bell, R., 1999, Distributed autonomous real-time planning and control of small to medium enterprises. *Proceedings of the Institute of Mechanical Engineers - Part B - Journal of Engineering Manufacture*, 213, 475-489.

Ramasesh, R., Kulkarni, S. and Jayakumar, M., 2001, Agility in manufacturing systems: an exploratory modeling framework and simulation. *Integrated Manufacturing Systems*, 12, 7, 534-548.

Ravindran, A., Phillips, D. T. and Solberg, J. J., 1987, *Operations research principles and practice,* 2nd Ed. (Wiley: New York).

Rigby, C., Day, M., Forrester, P. and Burnett, J., 2000, Agile supply: rethinking systems thinking, systems practice. *International Journal of Agile Management Systems*, 2, 3, 178-186.

Ryu, K. and Jung, M., 2002, Dynamic modeling of fractal-specific characteristics in fractal manufacturing systems. *Proceedings of the 35th CIRP international seminar on manufacturing systems*, Seoul, Korea, 12-15 May.

Ryu, K. and Jung, M., 2003, Agent-based fractal architecture and modelling for developing distributed manufacturing systems. *International Journal of Production Research*, 41, 17, 4233-4255.

Ryu, K. and Jung, M., 2004, Goal-orientation mechanism in a fractal manufacturing system. *International Journal of Production Research*, 42, 11, 2207-2225.

Ryu, K., Shin, M. and Jung, M., 2001, A methodology for implementing agent-based controllers in the fractal manufacturing system. *Proceedings of the 5th international conference on engineering design and automation (EDA 2001)*, Las Vegas, Nevada, USA, 5-8 August, pp.91-96.

Ryu, K., Shin, M., Kim, K. and Jung, M., 2000, Intelligent control architecture for fractal manufacturing system. *Proceedings of the 3rd Asia-Pacific conference on industrial engineering and management systems (APIEMS'2000)*, Hong Kong, 20-22 December, pp. 594-598.

Ryu, K., Son, Y. and Jung, M., 2003, Modeling and specifications of dynamic agents in fractal manufacturing systems. *Computers in Industry*, 52, 2, 161-182.

Saad, S. M., 2003, The reconfiguration issues in manufacturing systems. *Journal of Materials Processing Technology*, 138, 277-283.

Saad, S. M., Baykasoglu, A. and Gindy, N. N. Z., 2002a, A new integrated system for loading and scheduling in cellular manufacturing. *International Journal of Computer Integrated Manufacturing*, 15, 1, 37-49.

Saad, S. M., Baykasoglu, A. and Gindy, N. N. Z., 2002b, An integrated framework for reconfiguration of cellular manufacturing systems using virtual cells. *Production Planning & Control,* 13, 4, 381-393.

Saad, S. M., and Gindy, N. N., 1998, Handling internal and external disturbances in responsive manufacturing environments. *Production Planning & Control*, 9, 8, 760-770.

Saad, S. M. and Lassila, A. M., 2002, Sequencing for mixed-model assembly lines with bottleneck resources using tabu search. *Proceedings of the 12th International Conference on Flexible automation & Intelligent Manufacturing (FAIM2002),* 15-17 July, Dresden, Germany, pp. 664-673.

Sanchez, L. M. and Nagi, R., 2001, A review of agile manufacturing systems. *International Journal of Production Research,* 39, 16, 3561-3600.

Sarker, B. R. and Li, Z., 2001, Job routing and operations scheduling: a network-based virtual cell formation approach. *Journal of the Operational Research Society,* 52, 673-681.

Saunders, M., Lewis, P. and Thornhill, A., 2003, *Research methods for business students,* 3rd Ed. (Pearson Educational Ltd.: Essex).

Selladurai, V., Aravindan, P., Ponnambalam, S. G. and Gunasekaran, A., 1995, Dynamic simulation of job shop scheduling for optimal performance. *International Journal of Operations & Production Management,* 15, 7, 106-120.

Sharifi, H. and Zhang, Z., 1999, A methodology for achieving agility in manufacturing organisations: An introduction. *International Journal of Production Economics*, 62, 7-22.

Sharifi, H. and Zhang, Z., 2001, Agile manufacturing in practice - Application of a methodology. *International Journal of Operations & Production Management,* 21, 5/6, 772-794.

Sihn, W., 1995, Re-engineering through natural structures: the fractal factory. *Proceedings of SPIE - The International Society for Optical Engineers*, 2620, 62-69.

Sihn, W. and von Briel, R., 1997, Process cost calculation in a fractal company. *International Journal of Technology Management*, 13, 68-77.

Simsek, B. and Albayrak, S., 2003, Living factory: Back to Koestler in holonic manufacturing. *Proceedings of the INDIN-2003 workshop on innovative production machines and control systems*, Banff, Alberta, Canada, 21-24 August.

Sluga, A. and Butala, P., 2001, Self-organization in a distributed manufacturing system based on constraint logic programming. *Annals of the CIRP*, 50, 1, 323-326.

Small, A. W. and Downey, A. E., 1996, Orchestrating multiple changes: a framework for managing concurrent changes of varied type and scope. *Proceedings of the IEMC 1996 conference on managing virtual enterprise*, Canada, pp. 627-634.

Sousa, P. and Ramos, C., 1998, A dynamic scheduling holon for manufacturing orders. *Journal of Intelligent Manufacturing*, 9, 107-112.

Sousa, P. and Ramos, C., 1999, A distributed architecture and negotiation protocol for scheduling in manufacturing systems. *Computers in Industry*, 38, 103-113.

Sousa, P., Silva, N., Heikkila, T., Kollingbaum, M. and Valckenaers, P., 1999, Aspects of co-operation in distributed manufacturing systems. *Proceedings of the 2nd international workshop on intelligent manufacturing systems, IMS Europe'99*, Leuven, Belgium, 22-24 September.

Spear, S. and Bowen, H. K., 1999, Decoding the DNA of the Toyota production system. *Harvard Business Review*, 77, 5, 97-106.

Spiña, G., Bartezzaghi, E., Bert, A., Cagliano, R., Draaijer, D. and Boer, H., 1996, Strategically flexible production: the multi-focused manufacturing paradigm. *International Journal of Operations & Production Management*, 16, 11, 20-41.

St. John, C. H., Cannon, A. R. and Pouder, R. W., 2001, Change drivers in the new millennium: implications for manufacturing strategy research. *Journal of Operations Management,* 19, 143-160.

Strauss, R. E. and Hummel, T., 1995, The new industrial engineering revisited - Information technology, business process re-engineering, and lean management in the self-organizing "fractal company". *Proceedings of the 1995 IEEE annual international engineering management conference*, Singapore, June 1995, 1, 2, 287-292.

Suda, H., 1989, Future factory system formulated in Japan. *Techno Japan,* 22, 10, 15-25.

Suda, H., 1990, Future factory system formulated in Japan (2). *Techno Japan*, 23, 3, 51-61.

Sun, H. and Venuvinod, P. K., 2001, The human side of holonic manufacturing systems. *Technovation*, 21, 353-360.

Taylor, F. W., 1911, *The principles of scientific management.* (London: Harper&Brothers).

Tharumarajah, A., Wells, A. J. and Nemes, L., 1996, Comparison of the bionic, fractal and holonic manufacturing system concepts. *International Journal of Computer Integrated Manufacturing*, 9, 3, 217-226.

Tharumarajah, A., Wells, A. J. and Nemes, L., 1998, Comparison of emerging manufacturing concepts. *Proceedings of 1998 IEEE international conference of systems, man & cybernetics*, San Diego, CA, USA, 11-14 October, 1, 1, 325-331.

Tirpak, T. M., Daniel, S. M., Lalonde, J. D. and Davis, W. J., 1992, A note on a fractal architecture for modelling and controlling flexible manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 22, 3, 564-567.

Toh, K. T. K., Harding, J. A. and Thompson, D., 1999, An holonic approach for the modelling of enterprise functionality and behaviour. *International Journal of Computer Integrated Manufacturing*, 12, 6, 541-558.

Tsubone, H. and Horikawa, M., 1999, A comparison between machine flexibility and routing flexibility. *The International Journal of Flexible Manufacturing Systems*, 11, 83-101.

Ueda, K., 1992, A concept for bionic manufacturing systems based on DNA-type information. *Proceedings of the 8th international PROLAMAT conference: Human aspects in computer integrated manufacturing*, Tokyo, 853-863.

Ueda, K., Fujii, N. and Vaario, J., 1997a, Animation of biological manufacturing system. *Proceedings of the 8$^{th}$ DAAAM international symposium*, Dubrovnik, Croatia, 22-25 October.

Ueda, K., Fujii, N., Hatono, I. and Kobayashi, M., 2002, Facility layout planning using self-organization method. *Annals of the CIRP*, 51, 1, 399-402.

Ueda, K., Hatono, I., Fujii, N. and Vaario, J., 2000, Reinforcement learning approach to biological manufacturing systems. *Annals of the CIRP*, 49, 1, 343-346.

Ueda, K., Hatono, I., Fujii, N. and Vaario, J., 2001a, Line-less production system using self-organization: A case study for BMS. *Annals of the CIRP*, 50, 1, 319-322.

Ueda, K., Markus, A., Monostori, L., Kals, H. J. J. and Arai, T., 2001b, Emergent synthesis methodologies for manufacturing. *Annals of the CIRP*, 50, 2, 535-551.

Ueda, K., Vaario, J. and Fujii, N., 1998, Interactive manufacturing: Human aspects for biological manufacturing systems. *Annals of the CIRP*, 47, 1, 389-392.

Ueda, K., Vaario, J. and Ohkura, K., 1997b, Modelling of biological manufacturing systems for dynamic reconfiguration. *Annals of the CIRP*, 46, 1, 343-346.

Urban, T. L., Chiang, W. C. and Russell, R. A., 2000, The integrated machine allocation and layout problem. *International Journal of Production Research,* 38, 13, 2911-2930.

Vaario, J., 1994a, Modeling adaptive self-organization. *Proceedings of the 4th international workshop on the synthesis and simulation of living systems - Artificial life IV*, July, pp. 313-318.

Vaario, J., 1994b, Modeling biological adaptation. *Proceedings of the 1994 Japan-USA symposium on flexible automation*, July, pp. 1257-1262.

Vaario, J., 1996, Emergent scheduling based on the method of local attraction fields. *Proceedings of the third 'Biological engineering systems' group meeting*, Miyashima, Japan, 24-26 November.

Vaario, J., Fujii, N., Scheffter, D., Mezger, M. and Ueda, K., 1997, Factory animation by self-organization principles. *Proceedings of international conference on virtual systems and multimedia, VSMM'97*, Geneva, Switzerland, 10-12 September.

Vaario, J. and Ueda, K., 1996a, Self-organization in manufacturing systems. *Proceedings of the 1996 Japan-USA symposium on flexible automation*, Boston, MA, USA, 7-10 July.

Vaario, J. and Ueda, K., 1996b, Biological concept of self-organization in flexible automation systems. *Proceedings of the advanced product management systems conference (APM'96)*, Kyoto, Japan, 4-6 November, pp. 33-38.

Vaario, J. and Ueda, K., 1997, Biological concept of self-organization for dynamic shop-floor configuration. In *Advances in production management systems - Perspectives and future challenges* (Chapman & Hall).

Vaario, J. and Ueda, K., 1998a, An emergent modeling method for dynamic scheduling. *Journal of Intelligent Manufacturing*, 9, 2, 129-140.

Vaario, J. and Ueda, K., 1998b, Intelligent manufacturing by simple robots. *Proceedings of the third international symposium on artificial life and robotics (AROB III)*, pp. 93-96.

Valckenaers, P., van Brussel, H., Wyns, J., Bongaerts, L. and Peeters, P., 1998, Designing holonic manufacturing systems. *Robotics and Computer Integrated Manufacturing*, 14, 455-464.

van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, P., 1998, Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37, 255-274.

Venkatadri, U., Rardin, R. L. and Montreuil, B., 1997, A design methodology for fractal layout organization. *IIE Transactions*, 29, 911-924.

Vokurka, R. J. and O'Leary-Kelly, S. W., 2000, A review of empirical research on manufacturing flexibility. *Journal of Operations Management,* 18, 485-501.

Wadhwa, S. and Rao, K. S., 2003, Flexibility and agility for enterprise synchronization: Knowledge and innovation management towards flexagility. *Studies in Informatics and Control*, 12, 2, 111-128.

Wang, L., 2001, Integrated design-to-control approach for holonic manufacturing systems. *Robotics and Computer Integrated Manufacturing*, 17, 159-167.

Wang, K. J., Veeramani, D. and Chen, J., 2004, Distributed production planning using a graph-based nedotiation protocol. *International Journal of Production Research,* 42, 15, 3077-3099.

Wang, T. Y., Wu, K. B. and Liu, Y. W., 2001, A simulated annealing algorithm for facility layout problems under variable demand in cellular manufacturing systems. *Computers in Industry,* 46, 181-188.

Warnecke, H. J., 1993, *The fractal company - A revolution in corporate culture* (Berlin: Springer).

Warnecke, H. J. and Huser, M., 1995, Lean production. *International Journal of Production Economics*, 41, 37-43.

Warnock, I., 1996, *Manufacturing and business excellence: strategies, techniques and technology* (Englewood Cliffs, New Jersey: Prentice-Hall).

Wild, R., 1993, *Production and operations management.* 4th Ed. (Cassell: London)

Williams, M. and Vosniakos, G., 1996, On reference modelling of integrated manufacturing systems using OOA. *Proceedings of international conference on*

*design of information infrastructure systems for manufacturing DIISM'96*, Eindhoven, Netherlands, 15-18 September.

Williams, T. J., Bernus, P., Brosvic, J., Chen, D., Doumeingts, G., Nemes, L., Nevins, J. L., Vallespir, B., Vlietstra, J. and Zoetekouw, D., 1994, Architectures for integrating manufacturing activities and enterprises. *Computers in Industry,* 24, 111-139.

Willis, T. H., 1998, Operational competitive requirements for the twenty-first century. *Industrial Management & Data Systems*, 2, 83-86.

Womack, J. P., Jones, D. T. and Roos, D., 1990, *The machine that changed the world.* (New York: Rawson Associates).

Wu, N. and Zeng, W., 2002, Deadlock avoidance in an automated guidance vehicle system using a coloured Petri net model. *International Journal of Production Research*, 40, 1, 223-238.

Wullink, G., Giebels, M. M. T. and Kals, H. J. J., 2002, A system architecture for holonic manufacturing planning and control (EtoPlan). *Robotics and Computer Integrated Manufacturing*, 18, 313-318.

Wyns, J., 1999, *Reference Architecture for holonic manufacturing systems - The key to support evolution and reconfiguration.* Ph.D. thesis (Katholieke Universiteit Leuven, Leuven, Belgium).

Wyns, J., van Brussel, H., Valckenaers, P. and Bongaerts, L., 1996, Workstation architecture in holonic manufacturing systems. *Proceedings of the 28th CIRP international seminar on manufacturing systems,* Johannesburg, South Africa, 15-17 May, pp. 220-231.

Yang, B., Burns, N. D. and Backhouse, C. J., 2004, Management of uncertainty through postponement. *International Journal of Production Research*, 42, 6, 1049-1064.

Yapa, S. T. W. S., 2003, *A structured approach to rapid simulation model development.* Ph.D. thesis (Sheffield Hallam University, Sheffield, UK).

Yeh, C. H., 2000, A customer-focused planning approach to make-to-order production. *Industrial Management and Data Systems*, 4, 180-187.

Yusuf, Y. Y., Adeleye, E. O. and Sivayoganathan, K., 2003, Volume flexibility: the agile manufacturing conundrum. *Management Decision*, 41, 7, 613-624.

Yusuf, Y. Y., Sarhadi, M. and Gunasekaran, A., 1999, Agile manufacturing: the drivers, concepts and attributes. *International Journal of Production Economics*, 62, 33-43.

Zhang, Q., Vonderembse, M., A. and Lim, J. S., 2003, Manufacturing flexibility: defining and analysing relationships among competence, capability, and customer satisfaction. *Journal of Operations Management*, 21, 173-191.

# Appendices

## Appendix A: Responsive methodology modelled in Siman language

```
;   Model statements for module:  Create 11
;
49$         CREATE,       25,HoursToBaseTime(0.0),Entity 1:HoursToBaseTime(40),1:NEXT(50$);

50$         ASSIGN:       Create Product 1.NumberOut=Create Product 1.NumberOut + 1:NEXT(0$);
;
;   Model statements for module:  Assign 2
;
0$          ASSIGN:       Product Index=1:
                          Step Number=1:
                          Next Requirement=DNA(Product Index):
                          Entity.Picture=Product Pictures(Product Index):
                          Processing Time=Times(Product Index):
                          Due Date=100:NEXT(10$);
;
;   Model statements for module:  Decide 22
;
10$         BRANCH,       1:
                          If,Next Requirement==1,30$,Yes:
                          If,Next Requirement==2,33$,Yes:
                          If,Next Requirement==3,36$,Yes:
                          If,Next Requirement==4,39$,Yes:
                          If,Next Requirement==5,42$,Yes:
                          Else,28$,Yes;
;
;   Model statements for module:  Assign 43
;
28$         ASSIGN:       Number out=Number out+1:NEXT(29$);
;
;   Model statements for module:  Dispose 1
;
29$         ASSIGN:       Dispose 1.NumberOut=Dispose 1.NumberOut + 1;
55$         DISPOSE:      Yes;
;
;   Model statements for module:  Hold 1
;
30$         QUEUE,        Hold 1.Queue;
            SCAN:         NQ(Seize 1.Queue)==0 || NQ(Seize 2.Queue) == 0:NEXT(32$);
;
;   Model statements for module:  Decide 23
;
32$         BRANCH,       1:
                          If,NQ(Seize 1.Queue)==0,56$,Yes:
                          Else,57$,Yes;
56$         ASSIGN:       Decide 23.NumberOut True=Decide 23.NumberOut True + 1:NEXT(11$);

57$         ASSIGN:       Decide 23.NumberOut False=Decide 23.NumberOut False + 1:NEXT(15$);
;
;   Model statements for module:  Seize 1
;
11$         QUEUE,        Seize 1.Queue;
            SEIZE,        2,Other:
                          SELECT(Machine1,POR, Machine copy),1:NEXT(59$);

59$         DELAY:        0.0,,VA:NEXT(13$);
```

```
;
;   Model statements for module:  Process 2
;
13$         ASSIGN:       Process 2.NumberIn=Process 2.NumberIn + 1:
                          Process 2.WIP=Process 2.WIP+1;
61$         DELAY:        Processing Time,,VA;
108$        ASSIGN:       Process 2.NumberOut=Process 2.NumberOut + 1:
                          Process 2.WIP=Process 2.WIP-1:NEXT(14$);

;   Model statements for module:  Release 1
;
14$         RELEASE:      Machine1(Machine copy),1:NEXT(27$);
;
;   Model statements for module:  Assign 42
;
27$         ASSIGN:       Step Number=Step Number+1:
                          Next Requirement=DNA(Product Index):
                          Entity.Picture=Product Pictures(Product Index):
                          Processing Time=Times(Product Index):NEXT(10$);
;
;   Model statements for module:  Seize 2
;
15$         QUEUE,        Seize 2.Queue;
            SEIZE,        2,Other:
                          SELECT(Machine2,POR, Machine copy),1:NEXT(112$);

112$        DELAY:        0.0,,VA:NEXT(17$);
;
;   Model statements for module:  Process 3
;
17$         ASSIGN:       Process 3.NumberIn=Process 3.NumberIn + 1:
                          Process 3.WIP=Process 3.WIP+1;
114$        DELAY:        Processing Time,,VA;
161$        ASSIGN:       Process 3.NumberOut=Process 3.NumberOut + 1:
                          Process 3.WIP=Process 3.WIP-1:NEXT(18$);
;
;   Model statements for module:  Release 2
;
18$         RELEASE:      Machine2(Machine copy),1:NEXT(27$);
;
;   Model statements for module:  Hold 2
;
33$         QUEUE,        Hold 2.Queue;
            SCAN:         NQ(Seize 2.Queue)==0 || NQ(Seize 3.Queue) == 0:NEXT(35$);
;
;   Model statements for module:  Decide 24
;
35$         BRANCH,       1:
                          If,NQ(Seize 2.Queue)==0,164$,Yes:
                          Else,165$,Yes;
164$        ASSIGN:       Decide 24.NumberOut True=Decide 24.NumberOut True + 1:NEXT(15$);

165$        ASSIGN:       Decide 24.NumberOut False=Decide 24.NumberOut False + 1:NEXT(19$);
;
;   Model statements for module:  Seize 3
;
19$         QUEUE,        Seize 3.Queue;
            SEIZE,        2,Other:
                          SELECT(Machine3,POR, Machine copy),1:NEXT(167$);
```

```
167$        DELAY:        0.0,,VA:NEXT(21$);
;
;   Model statements for module:  Process 4
;
21$         ASSIGN:       Process 4.NumberIn=Process 4.NumberIn + 1:
                          Process 4.WIP=Process 4.WIP+1;
169$        DELAY:        Processing Time,,VA;
216$        ASSIGN:       Process 4.NumberOut=Process 4.NumberOut + 1:
                          Process 4.WIP=Process 4.WIP-1:NEXT(22$);
;
;   Model statements for module:  Release 3
;
22$         RELEASE:      Machine3(Machine copy),1:NEXT(27$);
;
;   Model statements for module:  Hold 3
;
36$         QUEUE,        Hold 3.Queue;
            SCAN:         NQ(Seize 3.Queue)==0 || NQ(Seize 4.Queue) == 0:NEXT(38$);
;
;   Model statements for module:  Decide 25
;
38$         BRANCH,       1:
                          If,NQ(Seize 3.Queue)==0,219$,Yes:
                          Else,220$,Yes;
219$        ASSIGN:       Decide 25.NumberOut True=Decide 25.NumberOut True + 1:NEXT(19$);

220$        ASSIGN:       Decide 25.NumberOut False=Decide 25.NumberOut False + 1:NEXT(45$);
;
;   Model statements for module:  Seize 4
;
45$         QUEUE,        Seize 4.Queue;
            SEIZE,        2,Other:
                          SELECT(Machine4,POR, Machine copy),1:NEXT(222$);

222$        DELAY:        0.0,,VA:NEXT(47$);
;
;   Model statements for module:  Process 5
;
47$         ASSIGN:       Process 5.NumberIn=Process 5.NumberIn + 1:
                          Process 5.WIP=Process 5.WIP+1;
224$        DELAY:        Processing Time,,VA;
271$        ASSIGN:       Process 5.NumberOut=Process 5.NumberOut + 1:
                          Process 5.WIP=Process 5.WIP-1:NEXT(48$);
;
;   Model statements for module:  Release 4
;
48$         RELEASE:      Machine4(Machine copy),1:NEXT(27$);
;
;   Model statements for module:  Hold 4
;
39$         QUEUE,        Hold 4.Queue;
            SCAN:         NQ(Seize 4.Queue)==0 || NQ(Seize 5.Queue) == 0:NEXT(41$);
;
;   Model statements for module:  Decide 26
;
41$         BRANCH,       1:
                          If,NQ(Seize 4.Queue)==0,274$,Yes:
                          Else,275$,Yes;
274$        ASSIGN:       Decide 26.NumberOut True=Decide 26.NumberOut True + 1:NEXT(45$);
```

```
275$        ASSIGN:       Decide 26.NumberOut False=Decide 26.NumberOut False + 1:NEXT(23$);
;
;   Model statements for module:  Seize 5
;
23$         QUEUE,        Seize 5.Queue;
            SEIZE,        2,Other:
                          SELECT(Machine5,POR, Machine copy),1:NEXT(277$);

277$        DELAY:        0.0,,VA:NEXT(25$);
;
;   Model statements for module:  Process 6
;
25$         ASSIGN:       Process 6.NumberIn=Process 6.NumberIn + 1:
                          Process 6.WIP=Process 6.WIP+1;
279$        DELAY:        Processing Time,,VA;
326$        ASSIGN:       Process 6.NumberOut=Process 6.NumberOut + 1:
                          Process 6.WIP=Process 6.WIP-1:NEXT(26$);

;   Model statements for module:  Release 5
;
26$         RELEASE:      Machine5(Machine copy),1:NEXT(27$);
;
;   Model statements for module:  Hold 5
;
42$         QUEUE,        Hold 5.Queue;
            SCAN:         NQ(Seize 5.Queue)==0 || NQ(Seize 1.Queue) == 0:NEXT(44$);
;
;   Model statements for module:  Decide 27
;
44$         BRANCH,       1:
                          If,NQ(Seize 5.Queue)==0,329$,Yes:
                          Else,330$,Yes;
329$        ASSIGN:       Decide 27.NumberOut True=Decide 27.NumberOut True + 1:NEXT(23$);

330$        ASSIGN:       Decide 27.NumberOut False=Decide 27.NumberOut False + 1:NEXT(11$);
;
;   Model statements for module:  Create 12
;
331$        CREATE,       15,HoursToBaseTime(0.0),Entity 2:HoursToBaseTime(40),1:NEXT(332$);

332$        ASSIGN:       Create Product 2.NumberOut=Create Product 2.NumberOut + 1:NEXT(1$);
;
;   Model statements for module:  Assign 3
;
1$          ASSIGN:       Product Index=2:
                          Step Number=1:
                          Next Requirement=DNA(Product Index):
                          Entity.Picture=Product Pictures(Product Index):
                          Processing Time=Times(Product Index):
                          Due Date=100:NEXT(10$);
;
;   Model statements for module:  Create 13
;
335$        CREATE,       40,HoursToBaseTime(0.0),Entity 3:HoursToBaseTime(40),1:NEXT(336$);

336$        ASSIGN:       Create Product 3.NumberOut=Create Product 3.NumberOut + 1:NEXT(2$);
;
;   Model statements for module:  Assign 4
2$          ASSIGN:       Product Index=3:
                          Step Number=1:
```

```
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);
;
;   Model statements for module:  Create 14
;
339$        CREATE,       10,HoursToBaseTime(0.0),Entity 4:HoursToBaseTime(40),1:NEXT(340$);

340$        ASSIGN:       Create Product 4.NumberOut=Create Product 4.NumberOut + 1:NEXT(3$);
;
;   Model statements for module:  Assign 5
;
3$          ASSIGN:       Product Index=4:
                Step Number=1:
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);
;
;   Model statements for module:  Create 15
;
343$        CREATE,       15,HoursToBaseTime(0.0),Entity 5:HoursToBaseTime(40),1:NEXT(344$);

344$        ASSIGN:       Create Product 5.NumberOut=Create Product 5.NumberOut + 1:NEXT(4$);
;
;   Model statements for module:  Assign 18
;
4$          ASSIGN:       Product Index=5:
                Step Number=1:
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);
;
;   Model statements for module:  Create 16
;
347$        CREATE,       15,HoursToBaseTime(0.0),Entity 6:HoursToBaseTime(40),1:NEXT(348$);

348$        ASSIGN:       Create Product 6.NumberOut=Create Product 6.NumberOut + 1:NEXT(5$);
;
;   Model statements for module:  Assign 19
;
5$          ASSIGN:       Product Index=6:
                Step Number=1:
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);
;
;   Model statements for module:  Create 17
;
351$        CREATE,       30,HoursToBaseTime(0.0),Entity 7:HoursToBaseTime(40),1:NEXT(352$);

352$        ASSIGN:       Create Product 7.NumberOut=Create Product 7.NumberOut + 1:NEXT(6$);
;
;   Model statements for module:  Assign 20
;
6$          ASSIGN:       Product Index=7:
                Step Number=1:
```

```
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);
;
;   Model statements for module:  Create 18
;
355$        CREATE,      5,HoursToBaseTime(0.0),Entity 8:HoursToBaseTime(40),1:NEXT(356$);

356$        ASSIGN:      Create Product 8.NumberOut=Create Product 8.NumberOut + 1:NEXT(7$);
;
;   Model statements for module:  Assign 21
;
7$          ASSIGN:      Product Index=8:
                Step Number=1:
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);
;
;   Model statements for module:  Create 19
;
359$        CREATE,      30,HoursToBaseTime(0.0),Entity 9:HoursToBaseTime(40),1:NEXT(360$);

360$        ASSIGN:      Create Product 9.NumberOut=Create Product 9.NumberOut + 1:NEXT(8$);
;
;   Model statements for module:  Assign 34
;
8$          ASSIGN:      Product Index=9:
                Step Number=1:
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);

;   Model statements for module:  Create 20
;
363$        CREATE,      15,HoursToBaseTime(0.0),Entity 10:HoursToBaseTime(40),1:NEXT(364$);

364$        ASSIGN:      Create Product 10.NumberOut=Create Product 10.NumberOut + 1:NEXT(9$);
;
;   Model statements for module:  Assign 35
;
9$          ASSIGN:      Product Index=10:
                Step Number=1:
                Next Requirement=DNA(Product Index):
                Entity.Picture=Product Pictures(Product Index):
                Processing Time=Times(Product Index):
                Due Date=100:NEXT(10$);
```

# Appendix B: Fractal layout design methodology modelled in C++ language

Fractal.h

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#ifndef FRACTAL_H
#define FRACTAL_H

#define LIMIT 20                    // maximum internal size of cells 20x20
#define MAXF 11                     // maximum number of cells
#define MAXMT 6                     // maximum number of different machine types
#define PERIOD 100
#define MU 0.9
#define RANDCASES 100000
#define STATFILE "output.txt"

// GLOBAL DEFINITIONS
enum compositions {none,identical,similar,minimal,idopt,optimal,minopt,special};
enum direction {left,right,up,down};

#include "tabu.h"
#include "products.h"

void showoptions();
void assdem(int, const Product*, Cells*);
void assmacheven(int, int, int*, Cells*);
void assdemopt(int,int,int,const Product*,Cells*,compositions,bool sample=false);
void assdemminopt(int, int, int, const Product*, Cells*);
void forceidentical(int,Cells*);
int printcaseeval(int,int,const Cells*,bool,int specialmt=-1);  // print evaluation of case

class Location{             // all inline
public:
        int a;
        int b;
        void operator =(const Location& source)
                { a = source.a; b = source.b; }
        friend bool operator ==(const Location& x, const Location& y)
                { return (x.a == y.a && x.b == y.b); }
};

struct cases {
        int amount;
        int results;
        int randres;
        double sdev;
        long unsigned int visits;
};

struct traffic {
        char mtype;
        int pt; // process time if ingress, product type if egress
        direction edge;
        Location peer;
        Location cmachpos;
        bool done;
};
```

```
class Routes{                   // all inline
public:
        Routes() { for (int i=0; i<MAXF; i++) amount[i] = 0; }
        int amount[MAXF];
};

struct machine{
        char type;
        int avail;
};

#include "problems.h"

class Cells{
public:
        // basic functions
        Cells();
        ~Cells();
        void clearcell();
        void operator =(const Cells& source);

        // position & size
        void setpos(int,const Location*);
        void getcellpos(Location& x) const { x.a = pos.a; x.b = pos.b; }
        int getsizea() const { return size.a; }
        int getsizeb() const { return size.b; }
        int neighsize() const;

        // product demand
        void adddemand(int,int,const Product*);
        void rmdemand(int,int,const Product*);
        void cleardemand();
        int getdemand(int ptype) const { return demand[ptype]; }

        // machine demand
        int getmachdem(int mtype) const { return machdem[mtype]; }
        void decmachdem(int mtype, int amount) { machdem[mtype] -= amount; }
        void resetmachdem(const Product*);
        bool ability(bool special = false, int specialmt=-1) const;
        int getexcess(int, int* n=NULL, bool special = false, int specialmt=-1) const;
        bool routeprod(int,int,bool,int period=PERIOD);

        // machines
        int gettotalmach() const { return totalmach; }
        void addmach(int,int);
        void rmmach(int,int);
        int getmnum(int mtype) const { return machnum[mtype]; }
        int findmach(char,int,Location*) const;

        // permutation
        int makepermut(Permutation*) const;
        void writepermut(int,const Permutation*,int period=PERIOD);
        void randomise();
        void quickdraw(bool disk=false,FILE *outf=NULL) const;
        int simulate(const Product*);

        // problems
        int routed(machine*, int) const;
        int countr(int,Problems*,const Product*) const;
```

```
        // external routing
        int ingress;                    // number of products externally coming in
        int egress;                             // number of products leaving
        void allocingress() { in = new traffic[ingress]; }
        void allocegress() { out = new traffic[egress]; }
        void addingress(int,char,Location*);
        void addegress(int,char,Location*);
        void resettraf(bool);
        void setdone(bool,int);
        void copytraf(bool,int,traffic&) const;

private:
        void setsize(int);
        Location pos;           // position of cell on shop floor
        Location size;          // internal size of cell
        int demand[MAXPT];      // demand in cell for each product
        int machnum[MAXMT];     // number of machines of type in cell
        int machdem[MAXMT];     // processing demand on each machine type
        int totalmach;
        machine m[LIMIT][LIMIT];
        traffic* in;
        traffic* out;
};

void calcsq(int, Location*);            // get dimensions of shop floor
void getpos(int, const Location*, Location*);
void rsequence(Location*,Location*,int,int*&,Cells* c);
int evalroutes(int, int, Cells*, const Product*,bool,int);
int optextpos(int, int, Cells*, const Product*,int period=PERIOD);
void countexta(Cells*,int,int,int,int&,int&);
int countextir(int, Cells*);
int findcell (int, Cells*, const Location*);
void msort(int,int*&,const int*);
#endif
```

Fractal.cpp

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#include "fractal.h"
#include <stdio.h>
#include <assert.h>
#include <time.h>
#include <stdlib.h>

// compositions can be identical,similar,minimal,idopt,optimal,minopt,special

int main(int argc, char *argv[ ]){
        compositions comp;
        int i, j, total, f, currentf, period;
        int inttime = 0, exttime = 0, extitime = 0, extrtime = 0, extritime = 0;

        // Initialisations
        srand((unsigned)time(NULL));

        // read product demand and sequence
        Product P[MAXPT];
        int maxp = P->readprods();
```

```
if(maxp == 0) return 1;     // zero value is error in readprods

// get option from user
comp = none;
if (argc > 1) i = strtol(argv[1],NULL,10);
comp = compositions(i);
if ( comp < 1 || comp > 7 ) {
        showoptions();
        do {
                printf("\nEnter the desired option -> ");
                scanf("%i",&comp);
        } while ( comp < 1 || comp > 7 );
}

// calculate required machines
int nummt[MAXMT];       // number of machines required of each type
int maxm = P->getmtdem(maxp,&nummt[0]);
assert(maxm <= MAXMT);

// define number of cells
if (argc > 2) f = strtol(argv[2],NULL,10);
else {
        do {
                printf("\nEnter the number of cells [max %i] -> ",MAXF);
                scanf("%i",&f);
        } while (f<1 || f>MAXF);
}
assert(f>0 && f<=MAXF);

Location lim;
calcsq(f,&lim);
if (f == 1) { comp = similar; printf("\nSetting method to similar"); } // for safety

char layoutfile[12] = "lyt___.txt";
layoutfile[3] = char(int(comp))+'0';
layoutfile[4] = char(f/10)+'0';
layoutfile[5] = char(f%10)+'0';

FILE *outf;
outf = fopen(layoutfile,"w");

// create cells
Cells* c = new Cells[f];
for (i=0; i<f; i++) c[i].setpos(i,&lim);
bool specialdone = false;
int specialmt = -1;

// Add machines and demand according to option

if ( comp == identical || comp == similar ) {
        assdem(f, P, c); // allocate demand to cells
        // calculate required machines per cell
        for (i=0; i<f; i++) {
                P->getmtdem(maxp,&nummt[0],&c[i]);
                for (j=0; j<maxm; j++)
                        c[i].addmach(j,nummt[j]);
                //c[i].quickdraw();
        }
}
```

```
        if ( comp == minimal || comp == minopt || comp == optimal || comp == idopt )
                assmacheven(f, maxm, nummt, c);

        if ( comp == identical || comp == idopt ) forceidentical(f,c);

        if ( comp == special ) {
                total = 0;
                for (i=0; i<maxm; i++) total += nummt[i];
                int* snummt = new int[maxm];
                msort(maxm,snummt,&nummt[0]);
                for (i=0; i<maxm; i++)
                        if (nummt[snummt[i]] < f-1) {
                                j = int((total-nummt[snummt[i]]) / (f-1)); if ((total-
nummt[snummt[i]]) % (f-1) > 0) j++;
                                if (c[f-1].gettotalmach() + nummt[snummt[i]] <= j + 2) // wouldn't be
too big
                                {
                                        c[f-1].addmach(snummt[i],nummt[snummt[i]]);
                                        total -= nummt[snummt[i]];
                                        specialmt = snummt[i];
                                        nummt[snummt[i]] = 0;
                                        specialdone = true;
                                }
                        }
                // assign the rest
                if (specialdone == true) {
                        assmacheven(f-1,maxm,nummt, c);
                        printf("Created specialised cell\n");
                }
                else {
                        assmacheven(f,maxm,nummt, c);
                        printf("Couldn't create specialised cell from list of required machines\n");
                }
                // delete [] snummt;  // line will crash - dunno why
        }

        if ( comp == optimal || comp == idopt || comp == special ) {
                if (specialdone == true) {
                        for (i=0; i<maxm; i++)
                                if (c[f-1].getmnum(i) > 0)
                                        for (j=0; j<f-1; j++)
                                                c[j].addmach(i,c[f-1].getmnum(i));
                        assdemopt(f-1,maxp,maxm,P,c,special);
                        for (i=0; i<maxm; i++)
                                if (c[f-1].getmnum(i) > 0)
                                        for (j=0; j<f-1; j++)
                                                c[j].rmmach(i,c[f-1].getmnum(i));
                }
                else assdemopt(f,maxp,maxm,P,c,optimal);

                if ( comp == idopt ) forceidentical(f,c);
                for (i=0; i<f; i++) {
                        //c[i].quickdraw();
                        //for(j=0;    j<maxp;    j++)    printf("prod    %i    has    demand    of
%i\n",j,c[i].getdemand(j));
                }
        }

        if ( comp == minimal || comp == minopt || comp == special) {
                if ( comp == minopt ) assdemopt(f,maxp,maxm,P,c,comp);
```

```
                if ( comp == minimal ) assdem(f, P, c);
                if (f > 2)
                        extrtime = 2 * optextpos(f,maxm,c,P); // tabu search
                else
                        extrtime = 2 * evalroutes(f,maxm,c,P,true,PERIOD);

                Location x;
                for (i=0; i<f; i++) {
                        c[i].getcellpos(x);
                        //c[i].quickdraw();
                        //printf("\nLocation of cell %i: a=%i,b=%i",i,x.a,x.b);
                        //for(j=0;  j<maxp;  j++)  printf("\nprod  %i  has  demand  of
%i",j,c[i].getdemand(j));
//                      printf("\ningress: %i: ",c[i].ingress); printf("\negress: %i: ",c[i].egress);
                }
        }

        // Internal layout optimisation ==========================================
        for(currentf=0; currentf<f; currentf++) {
                printf("\nFractal number %i:",currentf);
                fprintf(outf,"\nFractal number %i:",currentf);
                inttime += tabusearch(P,&c[currentf],PERIOD);
                c[currentf].quickdraw(true,outf);
        }

        printf("\ntotal internal distance through all %i cells is: %i moves\n",f,inttime);
        fprintf(outf,"\ntotal internal distance through all %i cells is: %i moves\n",f,inttime);
        for (currentf = 0; currentf<f; currentf++)
                countexta(c,currentf,f,lim.a,exttime,extitime);
        extritime = countextir(f,c);
        printf("\ninput  output  routing  distance  is  %i  moves  equivalent  to  %i  internal
moves",exttime,extitime);
        fprintf(outf,"\ninput  output  routing  distance  is  %i  moves  equivalent  to  %i  internal
moves",exttime,extitime);
        printf("\nco-operative external routing distance is %i moves equivalent to %i internal
moves",extrtime,extritime);
        fprintf(outf,"\nco-operative external routing distance is %i moves equivalent to %i internal
moves",extrtime,extritime);
        total = 0; for (i=0; i<f; i++) total += c[i].egress;
        printf("\n%i co-operative routings took place, ",total);
        fprintf(outf,"\n%i co-operative routings took place, ",total);
        total = 0; for (i=0; i<f; i++) total += c[i].gettotalmach();
        printf("\n%i machines were on the shop floor",total);
        fprintf(outf,"\n%i machines were on the shop floor",total);
        printf("\ntotal  ext  distance  outside  cells  is  %i  moves  equivalent  to  %i  internal
moves\n",exttime+extrtime,extitime+extritime);
        fprintf(outf,"\ntotal ext distance outside cells is %i moves equivalent to %i internal
moves\n",exttime+extrtime,extitime+extritime);

        fclose(outf);
        if (comp == special) return 0;                  // flexibility test crashes for method 7
        if (argc > 2) return 0;                                 // no  flexibility  test  with  program
parameters

        // Resource capability testing for sample cases ==========================

        char statfile[12] = "out___.txt";
        statfile[3] = char(int(comp))+'0';
        statfile[4] = char(f/10)+'0';
        statfile[5] = char(f%10)+'0';
```

```
        printf("\n\nFlexibility tests can take a long time, i.e. hours");
        printf("\nOutput will be written to %s",statfile);

        char answer;
        do {
                scanf("%c",&answer);
                printf("\nRun flexibility test [Y/N] -> ");
                scanf("%c",&answer);
        } while (answer!='N' && answer!='Y');
        if (answer=='N') return 0;

        int iterations, ITERATIONS = 50;
        int values[9];
        Product RCase[MAXPT];
        total = 0;
        for (i=0; i<f; i++) total += c[i].gettotalmach();
        printf("\nCreating sample case...");
        outf = fopen(statfile,"w");
        // Print first lines to file
        fprintf(outf, "Method = %i - Fractals = %i - Total number of machines = %i\n\n", comp, f, total);
        fprintf(outf,"Sdiff\tMachVis\tTnone\tSnone\tTcoop\tScoop\tRcoop\tToptPd\tSoptPd\tRoptPd\n")
;
        // Run flexibility test
        for (int sdiff=0; sdiff <= 160; sdiff+=4) {
                for (i=0; i<9; i++) values[i] = 0;
                printf("\nSDIFF = %i\n",sdiff);
                for ( iterations = 0; iterations < ITERATIONS; iterations++ ) {
                        RCase->createcase(maxp,sdiff,P);
                        for(currentf=0; currentf<f; currentf++) c[currentf].cleardemand();      // also
clears machdem
                        values[0] += RCase->mvisited(maxp);
                        period = PERIOD;
                        if (comp == identical || comp == similar || comp == minimal) {
                                // MODIFYING ORIGINAL CELL CONFIGURATION !!!
                                assdem(f, RCase, c); // allocate demand to cells
                                if ( comp == identical || comp == similar ) {
                                        period = printcaseeval(f,maxm,c,false);
                                        values[1] += period;
                                        inttime = 0;
                                        for(currentf=0; currentf<f; currentf++) {

        c[currentf].writepermut(c[currentf].getsizea()*c[currentf].getsizeb(),NULL,period);
                                                c[currentf].ingress = 0; c[currentf].egress = 0;
                                                inttime += c[currentf].simulate(RCase);
                                        }
                                        printf("\nInternal routing distance = %i",inttime);
                                        values[2] += inttime;
                                        printf("\nTurning cooperation on...");
                                }
                                if (f > 1) {
                                        period = printcaseeval(f,maxm,c,true);
                                        values[3] += period;
                                        for(currentf=0; currentf<f; currentf++)

        c[currentf].writepermut(c[currentf].getsizea()*c[currentf].getsizeb(),NULL,period);
                                        extrtime = 2 * evalroutes(f,maxm,c,RCase,true,period);
                                        printf("\n%i product routings occurred",extrtime);
                                        values[5] += extrtime;
                                        inttime = 0;
```

```
                                        for(currentf=0; currentf<f; currentf++)
                                                inttime += c[currentf].simulate(RCase);
                                        printf("\nInternal routing distance = %i",inttime);
                                        values[4] += inttime;
                                        printf("\nTurning optimised distribution on...\n");
                                        for(currentf=0;         currentf<f;         currentf++)
c[currentf].cleardemand();
                                        assdemopt(f,maxp,maxm,RCase,c,comp,true);
                                        if ( comp != minimal ) {
                                                period = printcaseeval(f,maxm,c,false);
                                                values[6] += period;
                                        }
                                        for(currentf=0; currentf<f; currentf++)

        c[currentf].writepermut(c[currentf].getsizea()*c[currentf].getsizeb(),NULL,period);
                                        extrtime = 2 * evalroutes(f,maxm,c,RCase,true,period);
                                        printf("\n%i product routings occurred",extrtime);
                                        values[8] += extrtime;
                                        inttime = 0;
                                        for(currentf=0; currentf<f; currentf++)
                                                inttime += c[currentf].simulate(RCase);
                                        printf("\nInternal routing distance = %i",inttime);
                                        values[7] += inttime;
                                }
                        } else { // comp == idopt, optimal, minopt, special
                                // MODIFYING ORIGINAL CELL CONFIGURATION !!!
                                if ( comp == special ) {
                                        c[f-1].resetmachdem(P);
                                        assdemopt(f-1,maxp,maxm,RCase,c,comp,true);
                                }
                                else
                                        assdemopt(f,maxp,maxm,RCase,c,comp,true);
                                period = printcaseeval(f,maxm,c,false,specialmt);
                                values[6] += period;
                                for(currentf=0; currentf<f; currentf++) {
                                        c[currentf].ingress = 0; c[currentf].egress = 0;

        c[currentf].writepermut(c[currentf].getsizea()*c[currentf].getsizeb(),NULL,period);
                                }

                                if ( comp == minopt ) evalroutes(f,maxm,c,RCase,true,period);

                                inttime = 0;
                                for(currentf=0; currentf<f; currentf++)
                                        inttime += c[currentf].simulate(RCase);
                                printf("Internal routing distance = %i\n",inttime);
                                values[7] += inttime;

                                if ( comp != minopt ) {
                                        printf("\nTurning cooperation on...");
                                        i = printcaseeval(f,maxm,c,true);
                                        values[3] += i;
                                        if (period > i) period = i;
                                }
                                extrtime = 2 * evalroutes(f,maxm,c,RCase,true,period);
                                printf("\n%i product routings occurred\n",extrtime);
                                values[8] += extrtime;
                                if ( comp != minopt ) {
                                        for(currentf=0; currentf<f; currentf++)
```

```
c[currentf].writepermut(c[currentf].getsizea()*c[currentf].getsizeb(),NULL,period);
                                        inttime = 0;
                                        for(currentf=0; currentf<f; currentf++)
                                                inttime += c[currentf].simulate(RCase);
                                        printf("Internal routing distance = %i\n",inttime);
                                        values[4] += inttime;
                                }
                        }
                        if (sdiff == 0) break;
                }
                fprintf(outf,"%i\t",sdiff);
                if (sdiff == 0)
                        for (i=0;i<9;i++) fprintf(outf,"%.2f\t",double(values[i]));
                else
                        for (i=0;i<9;i++) fprintf(outf,"%.2f\t",double(values[i])/ITERATIONS);
                fprintf(outf,"\n"); fflush(outf);
        }
        fclose(outf);


/*
        // Assess flexibility
        printf("\nDo you want to run a flexibility test (y/n)? -> ");
        char question;
        do { scanf("%c",&question); } while (question != 'y' && question != 'n');
        if (question == 'y') {
                cases samples[MAXPROD];
                int casepos;

                // create random cells
                Cells* randc = new Cells[f];
                for (i=0; i<f; i++) randc[i] = c[i];

                for (i=0; i<=MAXPROD; i++) {
                        samples[i].amount = 0; samples[i].results = 0;
                        samples[i].randres = 0;
                        samples[i].visits = 0;
                }
                // Product RCase[MAXPT];
                for (i=0; i<MAXPT; i++) RCase[i] = P[i];    // make copy of products

                for (i=0; i<=RANDCASES; i++) {
                        if (i % int(RANDCASES / 100) == 0) printf(".");
                        j = i % 140; if (j > 120) j = 1; if (j > 100) j = 99;                   //      emphasize
sides
                        casepos = RCase->createcapable(j,maxp,P);
                        samples[casepos].amount++;
                        //samples[casepos].visits += casemvistited(maxp,RCase);
                        samples[casepos].visits += RCase->mvisited(maxp);

                        // eval fixed distribution
                        for(currentf=0; currentf<f; currentf++) c[currentf].cleardemand();
                        assdem(f, RCase, c);
                        inttime = 0;
                        for(currentf=0; currentf<f; currentf++) {

        c[currentf].writepermut(c[currentf].getsizea()*c[currentf].getsizeb(),NULL);
                                inttime += c[currentf].simulate(RCase);
                        }
```

```
                        samples[casepos].results += inttime;

                        // eval random distribution
                        for (j=0; j<f; j++) randc[j].randomise();
                        for(currentf=0; currentf<f; currentf++) randc[currentf].cleardemand();
                        assdem(f, RCase, &randc[0]);
                        inttime = 0;
                        for(currentf=0; currentf<f; currentf++) {

        randc[currentf].writepermut(randc[currentf].getsizea()*randc[currentf].getsizeb(),NULL);
                                inttime += randc[currentf].simulate(RCase);
                        }
                        samples[casepos].randres += inttime;

                        // printf("\n%i: %i",i,inttime);
                        createcase(i,maxp,P,RCase);
                        printf("\nCase %i: ",i);
                        for (j=0; j<maxp; j++) printf("%i,",RCase[j].demand);
                        inttime = 0;
                        for (j=0; j<maxp; j++) inttime += RCase[j].demand;
                        printf(", %i products",inttime);

                }
                FILE *outf;
                outf = fopen(STATFILE,"w");
                fprintf(outf,"Dist.\tSamples\tS_opt\tS_rnd\tVisits");
                for (i=0;i<MAXPROD;i++) {
                        j = samples[i].amount;
                        if (j > 0) {
                                fprintf(outf,"\n%i\t%i\t%i\t%i",i*2,j,int(samples[i].results /
j),int(samples[i].randres / j));
                                fprintf(outf,"\t%f", double(samples[i].visits) / j );
                        }
                }
                fclose(outf);
                printf("\nOutput written to %s\n",STATFILE);
                delete [] randc;
        }
*/

        // clean up and return
        delete [] c;
        return 0;
}


// ==========================================================================
void assdem(int f, const Product* P, Cells* c){
        int i,j,k,l,min,minp,temp;
        // first evenly
        for (i=0; i<MAXPT; i++)
                for (j=0; j<f; j++)
                        c[j].adddemand(i,int(P[i].demand / f),P);

        // now leftovers
        for (i=0; i<MAXPT; i++)
                for (j=P[i].demand % f; j>0; j--) {
                        min=0; minp=0;
                        for (l=0; l<MAXPT; l++) min += c[0].getdemand(l);
                        for (k=1;k<f;k++) {
```

```
                                temp = 0;
                                for (l=0; l<MAXPT; l++) temp += c[k].getdemand(l);
                                if (temp<min) {
                                        min=temp; minp=k;
                                }
                        }
                        c[minp].adddemand(i,1,P);
                }
}

// ============================================================================
void assmacheven(int f, int maxm, int* nummt, Cells* c) {
        int i,j,l,k,min,minp,temp;
        // distribute machines except for leftovers
        int total = 0;
        for (i=0; i<maxm; i++) total += nummt[i];
        for (i=0; i<f; i++)
                for (j=0; j<maxm; j++)
                        c[i].addmach(j,int(nummt[j]/f));
        // now leftovers
        for (i=0; i<maxm; i++)
                for (j=nummt[i] % f; j>0; j--) {
                        min=0; minp=0;
                        for (l=0; l<maxm; l++) min += c[0].getmnum(l);
                        for (k=1;k<f;k++) {
                                temp = 0;
                                for (l=0; l<maxm; l++) temp += c[k].getmnum(l);
                                if (temp<min) {
                                        min=temp; minp=k;
                                }
                        }
                        c[minp].addmach(i,1);
                }
}

// ============================================================================
void assdemopt(int f, int maxp, int maxm, const Product* P, Cells* c, compositions comp, bool sample) {
        int i,j,k,l,m;
        int testmach;
        bool flag = false;
        Permutation pi;
        int total = pi.drand(maxp,P);
        dtabu(f,total,maxp,pi,P,c,comp);
        int* leftos;
        int* temp = new int[maxp];

        while (flag == false) {
                pi.distribute(f,total,P,c,comp);
                leftos = new int[pi.time];
                printf("\n%i leftovers",pi.time);
                if (pi.time == 0) flag = true;
                else { // no optimal solution found!
                        // find leftovers
                        for (j=0; j<maxp; j++) temp[j] = P[j].demand;
                        for (j = 0; j< f; j++) {
                                //printf("cell %i: ",j);
                                for (i = 0; i<maxp; i++) {
                                        temp[i] -= c[j].getdemand(i);
                                        //printf("%i,",c[j].getdemand(i));
                                }
```

```
                        //printf("\n");
                }
                k = 0;
                for (j=0; j<maxp; j++)
                        if(temp[j] > 0) {
                                for(i=0; i<temp[j]; i++) {
                                        leftos[k] = j;
                                        k++;
                                }
                        }
                if ((comp == optimal || comp == special) && sample == false) {
                        // check if adding 1 machine works
                        k = pi.time;
                        l = -1;
                        for(j=0; j<f; j++)
                                for(i=0; i<maxm; i++) {
                                        c[j].addmach(i,1);
                                        pi.distribute(f,total,P,c,comp);
                                        //fprintf(stream,"adding machine %c to cell %i
resulted in %i leftovers\n",'A'+i,j,pi.time);
                                        if ( pi.time < k ) {
                                                k = pi.time;
                                                l = j;
                                                testmach = i;
                                        }
                                        c[j].rmmach(i,1);
                                }
                        if (l != -1 ) {  // found improvement with 1 machine
                                printf("\nadding excess machine %c to cell
%i\n",'A'+testmach,l);
                                c[l].addmach(testmach,1);
                                pi.distribute(f,total,P,c,comp);
                        } else { // no improvement with just 1 machine
                                m=0xFFFFFFF;
                                for(j=0; j<f; j++) {
                                        k = 0;
                                        for (i=0; i<maxm; i++)
                                                k += c[j].getmachdem(i);
                                        if ( k < m) { m = k; l = j; }
                                }
                                k=0xFFFFFFF;
                                for (j=0; j<maxm; j++)
                                        if (c[l].getmnum(j) < k) {
                                                k = c[l].getmnum(j);
                                                testmach = j;
                                        }
                                printf("\nno improvement, but adding excess machine %c to
cell %i\n",'A'+testmach,l);
                                c[l].addmach(testmach,1);
                        }
                        //tabu search again unless done
                        if (pi.time > 0) dtabu(f,total,maxp,pi,P,c,comp);
                }
                if (comp == minopt || sample == true) {
                        // find out where to put leftover demand
                        int best;
                        if (comp == minopt) {
                                // Minimise external routing events
                                for (i=0; i<pi.time; i++) { // for each leftover
                                        best = 0xFFFFFFF;
```

```
                                        for (j=0; j<f; j++) {
                                                c[j].adddemand(leftos[i],1,P);
                                                k = c[j].routed(NULL,PERIOD);
                                                c[j].rmdemand(leftos[i],1,P);
                                                if (k < best) {
                                                        best = k;
                                                        l = j;
                                                }
                                        }
                                        c[l].adddemand(leftos[i],1,P);
                                }
                        } else {
                                // Minimise time unit excess
                                for (i=0; i<pi.time; i++) { // for each leftover
                                        best = 0xFFFFFFF;
                                        for (j=0; j<f; j++) {
                                                c[j].adddemand(leftos[i],1,P);
                                                if (best > c[j].getexcess(maxm)) {
                                                        best = c[j].getexcess(maxm);
                                                        l = j;
                                                }
                                                c[j].rmdemand(leftos[i],1,P);
                                        }
                                        c[l].adddemand(leftos[i],1,P);
                                }
                        }
                        flag = true; // escape from loop, we don't do tabu search again
                }
                delete [] leftos;
        }
    }
    delete [] temp;
}

// ============================================================================
void countexta(Cells* c,int currentf,int f,int lima,int& exttime,int& extitime) {
    int total = 0,i,j;
    Location src, transpos;
    int transit;
    c[currentf].getcellpos(src);
    for(i=0; i<MAXPT; i++) total += c[currentf].getdemand(i);

    for(i=0; i < src.a; i++) { // left cells same row
        exttime += total;
        for(j=0; j < f; j++) {
            c[j].getcellpos(transpos);
            if (transpos.a == i && transpos.b == src.b) transit = j;
        }
        extitime += c[transit].getsizea() * total;
    }
    for(i=src.a; i < lima-1; i++) { // right cells 1st row
        exttime += total;
        for(j=0; j < f; j++) {
            c[j].getcellpos(transpos);
            if (transpos.a == i && transpos.b == 0) transit = j;
        }
        extitime += c[transit].getsizea() * total;
    }
}
```

```
// ============================================================================
int countextir(int f, Cells* c) {
	int total =0,i,j,k,x;
	int dstcell;
	// first reset all traffic undone, count total routes
	for (i=0; i<f; i++) {
		c[i].resettraf(true);
		c[i].resettraf(false);
		total += c[i].egress;
	}
	if (total == 0) return 0;
	// create data structures
	struct paths {
		Location srcint;
		Location dstint;
		Location srcext;
		Location dstext;
	};
	paths* route = new paths[total];
	traffic stemp, dtemp;
	Location ltemp;
	// find route paths
	x = 0;
	for (i=0; i<f; i++)
		for (j=0; j<c[i].egress; j++) {
			c[i].copytraf(false,j,stemp);
			c[i].getcellpos(route[x].srcext);
			route[x].srcint = stemp.cmachpos;
			route[x].dstext = stemp.peer;
			dstcell = findcell(f,c,&stemp.peer);
			for(k=0; k<c[dstcell].ingress; k++) {
				c[dstcell].copytraf(true,k,dtemp); // copy ingress traffic from dstcell
				if(dtemp.done == false && dtemp.mtype == stemp.mtype &&
dtemp.peer == route[x].srcext) {
					route[x].dstint = dtemp.cmachpos;
					c[dstcell].setdone(true,k);
					break;
				}
			}
			//printf("\n%02.i) route product %i from machine %i:%i in cell %i:%i(%i) to
cell %i:%i(%i) for machine %c at
%i:%i",x,stemp.pt,route[x].srcint.a,route[x].srcint.b,route[x].srcext.a,route[x].srcext.b,i,route[x].dstext.a,r
oute[x].dstext.b,dstcell,stemp.mtype,route[x].dstint.a,route[x].dstint.b);
			x++;
		}
	// count travelling distances
	x = 0;
	for (i=0; i<total; i++) {
		// first cells between to be jumped over
		j = 1;
		while (route[i].srcext.a + j < route[i].dstext.a) { // left to right
			ltemp.a = route[i].srcext.a + j; ltemp.b = route[i].srcext.b;
			x += 2 * c[findcell(f,c,&ltemp)].getsizea();
			j++;
		}
		j = 1;
		while (route[i].srcext.a > j + route[i].dstext.a) { // right to left
			ltemp.a = route[i].dstext.a + j; ltemp.b = route[i].dstext.b;
			x += 2 * c[findcell(f,c,&ltemp)].getsizea();
			j++;
```

```
                }
                j = 1;
                while (route[i].srcext.b + j < route[i].dstext.b) { // up to down
                        ltemp.a = route[i].srcext.a; ltemp.b = route[i].srcext.b + j;
                        x += 2 * c[findcell(f,c,&ltemp)].getsizeb();
                        j++;
                }
                j = 1;
                while (route[i].srcext.b > j + route[i].dstext.b) { // down to up
                        ltemp.a = route[i].dstext.a; ltemp.b = route[i].dstext.b + j;
                        x += 2 * c[findcell(f,c,&ltemp)].getsizeb();
                        j++;
                }
                // then simple cross border distance
                x += 2; // it's one move in and out in any case
                if ( route[i].srcext.b == route[i].dstext.b ) // same row
                        x += 2 * abs( route[i].srcint.b - route[i].dstint.b );
                if ( route[i].srcext.a == route[i].dstext.a ) // same column
                        x += 2 * abs( route[i].srcint.a - route[i].dstint.a );
                if ( route[i].srcext.a != route[i].dstext.a && route[i].srcext.b != route[i].dstext.b ) {
                        if ( route[i].srcext.b < route[i].dstext.b ) { // diagonal from top
                                x += 2 * ( c[findcell(f,c,&route[i].srcext)].getsizeb() - route[i].srcint.b
+ route[i].dstint.b );
                        } else { // diagonal from bottom
                                x += 2 * ( route[i].srcint.b +
c[findcell(f,c,&route[i].dstext)].getsizeb() - route[i].dstint.b );
                        }
                }
                //printf("\n%02.i) increased moves to %i",i,x);
        }
        delete [] route;
        return x;
}

// ============================================================================

void msort(int max,int*& dst,const int* src){
// finds positions in src that the list would have if it was sorted
        int j, lowest, current = -1, x, pos = 0;
        while (pos <= max) {
                x = 0xFFFFFF;
                for (j=0; j<max; j++)
                        if ( src[j] > 0 && src[j] < x  && src[j] > current ) {
                                lowest = src[j];
                                x = lowest;
                        }
                current = lowest;
                for (j=0; j<max; j++)
                        if ( src[j] == current ) {
                                dst[pos] = j;
                                pos++;
                        }
        }
}

// ============================================================================

void forceidentical(int f,Cells* c) {
        int nummt[MAXMT];
        // get largest number of machine type in any cell
```

```
        int total = 0, i,j;
        for (i=0; i<MAXMT; i++) {
                nummt[i] = 0;
                for (j=0; j<f; j++) if(nummt[i] < c[j].getmnum(i)) nummt[i] = c[j].getmnum(i);
                total += nummt[i];
        }
        // add machines to make fractals identical
        for (i=0; i<f; i++)
                for (j=0; j<MAXMT; j++)
                        if(nummt[j] > c[i].getmnum(j))
                                c[i].addmach(j,nummt[j]-c[i].getmnum(j));
}

// ============================================================================

int printcaseeval(int f,int maxm,const Cells* c,bool coop,int specialmt) {
        // specialised cell present if specialmt is NOT default -1
        int n, resultn;
        int j = 0, result = -0x7FFFFFFF;
        double divresult;
        if (coop == false) {
                for(int currentf=0; currentf<f; currentf++) {
                        printf ("\n%i: ",currentf);
                        for (int i=0; i<maxm; i++)
                                printf ("%i, ", c[currentf].getmachdem(i));
                        if (specialmt != -1) j = c[currentf].getexcess(maxm,&resultn,true,specialmt);
                        else j = c[currentf].getexcess(maxm,&resultn);
                        printf ("excess by %i time units", j);
                        if (result < j && (specialmt == -1 || currentf != f-1)) result = j;
                }
                printf("\nSample requires %i time units more\n", result);
                result += PERIOD;
//              divresult = double(result) / double(resultn);
//              result = int(divresult);
//              if (divresult > result) result++;
//              printf("so the real result should be %i.\n", result+PERIOD);
        } else {
                for (int i=0; i<maxm; i++) {
                        j = 0; n = 0;
                        for(int currentf=0; currentf<f; currentf++) {
                                j += c[currentf].getmachdem(i);
                                n += c[currentf].getmnum(i);
                                //j -= int(c[currentf].getmnum(i) * MU * PERIOD);
                        }
                        divresult = double(j) / ( double(n) * MU);
                        j = int(divresult);
                        if (j < divresult) j++;
                        printf ("\n%i time units required on machine type %c", j, i+'A');
                        if (result < j) result = j;
                }
                printf("\nSample requires %i time units", result);
                // printf("\nso the real result should be %i.", result+PERIOD);
        }

        if (result < PERIOD) return PERIOD;
        return result;
}

// ============================================================================
```

```
void showoptions(){
        printf("\n 1. Similar product distribution, identical cell composition");
        printf("\n 2. Similar product distribution, similar cell composition");
        printf("\n 3. Similar product distribution, minimal cell composition");
        printf("\n 4. Different product distribution, identical cell composition");
        printf("\n 5. Different product distribution, optimal cell composition");
        printf("\n 6. Different product distribution, minimal cell composition");
        printf("\n 7. Different product distribution, specialised cell if possible\n");
}
```

Tabu.h

```
// Tabu Search Program by Anna-Maija Lassila
// Sheffield Hallam University, Sep. 2001
// Header file tabu.h

#ifndef TABU_H
#define TABU_H

#define MAXPROD 400                    // how many items maximal?
#define MAXMOVES 1800                  // total number of moves
#define MAXRMOVES 0                    // randomize moves when optimising product distribution
#define MAXNOIMP 15                    // max number of consecutive moves w/o improvement
#define TABUSIZE 10                    // length of tabu list
#define TABUCSIZE 2                    // size of tabu list when optimising external cell layout
#define TABURESTART 0                  // can start this many times from the same point

#include "buffer.h"
enum compositions;

class Cells;                           // cross-header-file prototypes
class Location;
class Product;
class Routes;

struct positions{                      // data type that will hold move positions
  int a;
  int b;
};

class Tabulist{                        // defining the tabu list data type
public:
  void list_clear() { a.clear_fifo(); b.clear_fifo(); }
  void list_push(positions& move)
                { a.push_fifo(TABUSIZE,move.a); b.push_fifo(TABUSIZE,move.b); }
  void list_pop() { a.pop_fifo(TABUSIZE); b.pop_fifo(TABUSIZE); }
  int list_size() { return a.getsize(); }
  bool list_find(positions&) const;

private:
  Buffer a;  // fifo buffer storing position A in an A->B swap
  Buffer b;  // fifo buffer storing position B in an A->B swap
};

class Permutation{        // defining the Permutation data type
public:
  void display(int) const; // displaying Permutation and time value
  int create_neigh(int,Permutation*,Cells*,Product*,int);  // creating neighbourhood
  void poison() { time *= 10; }      // make neighbour unattractive
```

```
  int makedneigh(int,int,Permutation*,Cells*,const Product*,compositions);
  int drand(int,const Product*);
  int cposneigh(int,int,Location*,Permutation*,Cells*,const Product*,int);
  void distribute(int,int,const Product*,Cells*,compositions);

  // Operators
  void operator =(const Permutation&);
  friend bool operator <(const Permutation& p1, const Permutation& p2)
    { return (p1.time < p2.time); }
  friend bool operator <=(const Permutation& p1, const Permutation& p2)
    { return (p1.time <= p2.time); }
  friend bool operator ==(const Permutation& p1, const Permutation& p2)
    { return (p1.time == p2.time); }
  friend positions compare(int, const Permutation&, const Permutation&);

  int time;          // time value of Permutation
  int perm[MAXPROD];  // list of integers to make up Permutation
};

// Non-Members
int find_best(int, int, Permutation*);    // find best value in neighbourhood table
int tabusearch(Product*,Cells*,int period);
void dtabu(int,int,int,Permutation&,const Product*,Cells*,compositions);
int dneighsize(int,int,const Product*);

#endif
```

Tabu.cpp

```
// Tabu Search Program by Anna-Maija Lassila
// Sheffield Hallam University, Sep. 2001
// Main program file tabu.cpp

#include <stdio.h>
#include <string.h>    // for strlen
#include <stdlib.h>    // for rand
#include <time.h>      // for time, needed for srand
#include "tabu.h"
#include "fractal.h"

int tabusearch(Product* P,Cells* c,int period){
        int neighlen, total;
        Permutation pi;          // current Permutation
        Permutation best;        // best Permutation found
        Permutation localbest;   // best Permutation in area
        int solution;            // best solution in neighbourhood
        int noimpmoves;                    // number of moves w/o improvement
        int restartmoves;        // how often did we go back to local best?
        positions move;          // pair of numbers showing swap move
        Tabulist tabu;           // declare tabu list
        bool allowed;            // 2 internal flag variables
        bool flag = true;        // true if just left a local best
        positions lbest_exit[TABURESTART+1];   // first moves after localbest
        int moves = 1;
        int tabusz = TABUSIZE;

        // Initialise tabu search
        total = c->makepermut(&pi);
        pi.time = c->simulate(&P[0]);
```

```
best = pi;
localbest = pi;
neighlen = c->neighsize();
if (neighlen == 0) moves = MAXMOVES; // don't do silly tabu search
if (tabusz > neighlen/4) tabusz = int(neighlen/4);
tabu.list_clear();          // empty tabu list

Permutation* neigh = new Permutation[neighlen];
noimpmoves = 0; restartmoves = 0;

//pi.display(total);
printf("\nnow optimising...");

while ( moves < MAXMOVES ) {
        moves++;
        c->makepermut(&pi);
        solution = pi.create_neigh(total,&neigh[0],c,&P[0],period);

        // === is this solution allowed? ==========================================
        do {
                move = compare(total,neigh[solution],pi);   // what move was made?
                allowed = true;                     // assume that move is allowed
                if (tabu.list_find(move))           // is move tabu?
                        if ( best <= neigh[solution] ) {    // aspiration criterium
                                neigh[solution].poison();       // make this move unattractive
                                // find solution again with previous best being poisoned
                                solution = find_best(solution,neighlen,&neigh[0]);
                                allowed = false;                // we have to repeat this
                        }
        } while(allowed == false);              // until move is allowed
        // === is this move just after a local best? ==============================
        if (flag == true) {
                lbest_exit[restartmoves] = move;
                flag = false;
        }
        // === is this a local best? ==============================================
        if (neigh[solution] < localbest) {    // new local best found
                localbest = neigh[solution];       // set local best
                flag = true;
                noimpmoves = 0;                 // be patient
                restartmoves = 0;               // allow all restart attempts
        } else noimpmoves++;                // otherwise lose some patience
        // === is this even the best? =============================================
        if (neigh[solution] < best)  {        // new best found
                best = neigh[solution];           // set new best
                printf(".");
                //c[currentf].quickdraw();
                //pi.display(total);
        }

        // === Did we exceed MAXNOIMP? ============================================
        if (noimpmoves > MAXNOIMP) {
                tabu.list_clear();              // clear tabu list
                noimpmoves = 0;                 // be patient again
                // === Restart, if restarting is still allowed ========================
                if (restartmoves < TABURESTART ) {
                        restartmoves++;             // Remember that we restarted
                        // fill the tabu list with lbest_exit and print it to screen
                        printf("\n=== RESTART No. %i === tabu: ", restartmoves);
                        for (int x = 0; x < restartmoves; x++ ) {
```

```
                                        // tabu move that left local best last time
                                        tabu.list_push(lbest_exit[x]);
                                        printf("(%i,%i) ",lbest_exit[x].a,lbest_exit[x].b);
                                }
                                //printf("\n");

                                // the move we are going to make should be appended to lbest_exit
                                flag = true;
                                pi = localbest;             // go back to local best
                                //printf("move %.3i: ", moves);
                                //pi.display(total);
                        }
                        // === Restarted too often, randomising ==================
                        else {
                                //printf("\n=== RANDOMISING ===\n");
                                //printf("move %.3i: ", moves);
                                c->randomise();
                                c->makepermut(&pi);
                                pi.time = c->simulate(&P[0]);
                                localbest = pi;             // reset localbest
                                restartmoves = 0;             // allow restarts again
                        }
                }
                // === MAXNOIMP not exceeded yet, making the move ============
                else {
                        pi = neigh[solution];             // make move
                        //printf("move %.3i: ", moves);
                        //printf("swapping %.2i with %.2i -> ",move.a,move.b);
                        //pi.display(total);

                        // === update tabu list ================================
                        if (tabusz > 0) {
                                if (tabu.list_size() == tabusz )
                                        tabu.list_pop();        // remove oldest move from tabu list
                                tabu.list_push(move);       // insert move into tabu list
                        }
                }
        }
        c->writepermut(total,&best,period);
        c->quickdraw();
        best.display(total);
        c->simulate(P); // to get traffic data right
        delete [] neigh;
        return best.time;
}

// Displaying Permutation and Permutation time to screen
void Permutation::display(int total) const{
  printf("distance %i - ", time);
  for ( int i=0; i<total; i++ ) printf("%c", perm[i]+'A');
  printf("\n");
}

// Creating neighbourhood and reporting position of best neighbour found
int Permutation::create_neigh(int total, Permutation* neigh, Cells* c, Product* P,int period){
  int x = 0;          // how many neighbours did we find?
  int i, j;
  int besttime = 0xFFFFFFF, bestx;
  for ( i = 0; i < total-1; i++) {
    for ( j = i+1; j < total; j++) {
```

```
                    if ( perm[i] != perm[j] ) {          // swap made a difference?
        for (int z = 0; z < total; z++)
                        neigh[x].perm[z] = perm[z];      // copy Permutation
                    neigh[x].perm[i] = perm[j];          // swap products
        neigh[x].perm[j] = perm[i];
                    c->writepermut(total,&neigh[x],period);
                    neigh[x].time = c->simulate(P);
        if (besttime > neigh[x].time) {
          besttime = neigh[x].time;
          bestx = x;
        }
        x++;
      }
    }
  }
  c->writepermut(total,&neigh[bestx],period);
  time = neigh[bestx].time;
  return bestx;
}

// Compare two Permutations and return swap move
// Information for inserting to or finding from Tabu list
positions compare(int total, const Permutation& p1, const Permutation& p2){
  int i = 0;
  positions move;
  move.a = -1; move.b = -1;
  while ( i < total && move.b < 0) {
    if ( p1.perm[i] != p2.perm[i] ) {
      if ( move.a < 0 ) move.a = i;
      else move.b = i;
    }
    i++;
  }
  return move;
}

// Overloaded = operator to assign Permutations
void Permutation::operator =(const Permutation& source){
  time = source.time;
  for (int i = 0; i < MAXPROD; i++) perm[i] = source.perm[i];
  return;
}

// Finding a certain move in the Tabu list
bool Tabulist::list_find(positions& move) const{
  int end = a.getsize();
  for ( int i = 0; i < end; i++ )
    if ( a.getprod(i) == move.a && b.getprod(i) == move.b )
      return true;
  return false;
}

// finding the best solution in the neighbourhood
int find_best(int start, int neighlen, Permutation* neigh){
        int best = start;
        int bests = 0;
        int* temp = new int[neighlen];
        for ( int i = 0; i < neighlen; i++ ) {
                if ( neigh[i] < neigh[best] ) {
                        best = i;
```

```
                        bests = 0;
                }
                else if ( neigh[i] == neigh[best] ) {
                        temp[bests] = i;
                        bests++;
                }
        }
        if (bests > 0) best = temp[rand() % bests];
        delete [] temp;
        return best;
}
```

Products.h

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#ifndef PRODUCTS_H
#define PRODUCTS_H

#define PRODFILE        "products.txt"      // prod input data file
#define MAXPT 12        // maximum number of different product types

#include "fractal.h"
#include "stdio.h"          // for "NULL" word

struct producttimes{                        // Single Processing instruction
        char type;                                  // Machine type
        int time;                               // Time at machine
};

class Product {
public:
        Product();                                                          // Clearing constructor
        int readprods();                                        // Read product data from file
        void operator =(const Product&);    // Copy operator
        unsigned int mvisited(int) const;    // sum over array of machs visited
        int createcapable(int,int,const Product*);      // Create a random case that can be processed
        void createcase(int,int,const Product*);            // create any case with constant D
        void randproctime();                                    // randomise processing times
        void randsequence();                                    // change processing sequence
        void printdata(int max=MAXPT) const;                // print product table (for debugging)
        int getmtdem(int maxp, int* nummt, const Cells* c=NULL) const;      // calculate Nm or Nmf

        int number;                                             // Type number, not really used
        int demand;                                             // Demand for this type
        int machneeded;                                 // Number of machines needed for processing
        producttimes ptime[MAXMT];      // Matrix of sequence and duration for processing
};

#endif
```

Products.cpp

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#include <stdio.h>
```

```
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "products.h"
#include "fractal.h"

// Read products from file
int Product::readprods(){
        FILE *pfile;
        char line[80];
        int lpos, length;
        int readlines = 0;
        int prod = 0;

        if((pfile = fopen(PRODFILE, "r")) == NULL) {
                fprintf(stderr, "Cannot open products file\n"); return(0); }
        while(fgets(line,80,pfile)) {
                length = strlen(line);
                readlines++;
                if(line[0]>='0' && line[0]<='9') {
                        if (prod>MAXPT) {
                                fprintf(stderr, "Too many different product types\n");
                                return 0;
                        }
                        // product number
                        lpos = 0; this[prod].number = 0;
                        do {
                                this[prod].number*=10;
                                this[prod].number+=line[lpos]-'0';
                                lpos++;
                        } while (line[lpos]>='0' && line[lpos]<='9');
                        while (lpos<=length && (line[lpos]<'0' || line[lpos]>'9')) lpos++;
                        // demand
                        this[prod].demand=0;
                        do {
                                this[prod].demand*=10;
                                this[prod].demand+=line[lpos]-'0';
                                lpos++;
                        } while (line[lpos]>='0' && line[lpos]<='9');
                        if (this[prod].demand<=0) {
                                fprintf(stderr, "Error in product file on line %i\n",readlines-1);
                                return (0);
                        }
                        // product times
                        this[prod].machneeded=0;
                        while (lpos<=length) {
                                if (line[lpos]>='A' && line[lpos]<='Z') {
                                        if(line[lpos]>'A'+MAXMT-1) {
                                                fprintf(stderr, "Too many machine types in product
file on line %i\n",readlines-1);
                                                return (0);
                                        }
                                        this[prod].ptime[this[prod].machneeded].type = line[lpos];
                                        lpos++;
                                        if(line[lpos]!=':') {
                                                fprintf(stderr, "Error in product file on line
%i\n",readlines-1);
                                                return (0);
                                        }
                                        lpos++;
```

```
                                        if(line[lpos]<'0' || line[lpos]>'9') {
                                                fprintf(stderr, "Error in product file on line
%i\n",readlines-1);
                                                return (0);
                                        }
                                        this[prod].ptime[this[prod].machneeded].time = 0;
                                        do {
                                                this[prod].ptime[this[prod].machneeded].time*=10;

        this[prod].ptime[this[prod].machneeded].time+=line[lpos]-'0';
                                                lpos++;
                                        } while (line[lpos]>='0' && line[lpos]<='9');
                                        this[prod].machneeded++;
                                }
                                else lpos++;
                        }
                        prod++;
                }
        }
        printf("Product file %s successfully read\n",PRODFILE);
        return prod;
}

// ============================================================================
// Constructor - resetting all values

Product::Product(){
        number = 0;
        machneeded = 0;
        demand = 0;
        for(int i=0; i<MAXMT; i++){
                ptime[i].type = ' ';
                ptime[i].time = 0;
        }
}

// ============================================================================
// Overloaded operator - duplicate values except for demand

void Product::operator =(const Product& src){
        int j;
        number = src.number;
        machneeded = src.machneeded;
        for (j=0; j<machneeded; j++) {
                ptime[j].time = src.ptime[j].time;
                ptime[j].type = src.ptime[j].type;
        }
}

// ============================================================================
// If cell == NULL (default) calculating all Nm and writing them into nummt, returning M
// Else calculates Nmf required in certain cell c

int Product::getmtdem(int maxp, int* nummt, const Cells* c) const{
        int machdem, maxm = 0;
        for(int i=0;i<MAXMT;i++) {
                machdem=0;
                for(int j=0; j<maxp; j++)
                        for(int k=0; k<this[j].machneeded; k++)
                                if(this[j].ptime[k].type == 'A'+i)
```

```
                                        if (c==NULL)
                                                machdem += this[j].demand*this[j].ptime[k].time;
                                        else
                                                machdem += c->getdemand(j) *
this[j].ptime[k].time;
                        nummt[i] = int(float(machdem)/(PERIOD*MU));
                        if(nummt[i]<float(machdem)/(PERIOD*MU)) nummt[i]++;
                        if(nummt[i]>0) maxm++;
                }
                return maxm;
}

// ============================================================================
// Calculate total number of machines visited over all products

unsigned int Product::mvisited(int maxp) const {
        unsigned int value = 0;
        for (int i=0; i<maxp; i++)
                value += this[i].machneeded * this[i].demand;
        return value;
}

// ============================================================================
// Create random case that is feasible to create
// similarity to case study is 0 to 100

int Product::createcapable(int similarity,int maxp,const Product* P) {
        int i,j,prod,addednum,total;
        bool excess;
        int nummt[MAXMT], testmt[MAXMT];      // machines required of each type
        int variance =0;

        // reset demand and find out D
        total = 0;
        for (i=0; i<MAXPT; i++) {
                this[i].demand = 0;
                total += P[i].demand;
        }

        // get limits from original case
        P->getmtdem(maxp,&nummt[0]);

        // Have similar starting condition
        i = 0;
        for (j=0; j<maxp; j++) {
                this[j].demand = int(similarity * P[j].demand / 100);
                i += this[j].demand;
        }

        // Fill up the rest
        while (i < total) {
                excess = false;
                prod = rand() % maxp;                 // pick random product
                addednum = 1 + rand() % 5;
                if (addednum > total-i) addednum = 1;
                this[prod].demand += addednum;
                i += addednum;

                this->getmtdem(maxp,&testmt[0]);
                for (j=0; j<MAXMT; j++) if (testmt[j] > nummt[j]) excess = true;
```

```
                if (excess == true) {
                        this[prod].demand -= addednum;
                        i -= addednum;
                        do { prod = rand() % maxp; } while (this[prod].demand <= 0);
                //      addednum = rand() % this[prod].demand;
                //      this[prod].demand -= addednum;
                //      i -= addednum;
                        this[prod].demand--;
                        i--;

                }
        }


        return (variance/2);
}

// ============================================================================
// Create random case with no feasibility check, constant D

void Product::createcase(int maxp, int sdiff, const Product* P) {
        int i, variance;
        int total;
        sdiff *= 2;         // so we don't have to devide variance by 2 all the time
        // copy demand and find out D
        total = 0;
        for (i=0; i<MAXPT; i++) {
                this[i].demand = P[i].demand;
                this[i] = P[i];
                total += P[i].demand;
        }
        if (sdiff < 1) return;
        do {
                do i = rand() % maxp; while ( this[i].demand < 1 );
                this[i].demand--;
                i = rand() % maxp;
                this[i].demand++;
                variance = 0;
                for (i=0; i<=maxp; i++)
                        variance += abs(P[i].demand - this[i].demand);
        } while (variance != sdiff);
}

// ============================================================================
// Randomise processing times (not working yet)

void Product::randproctime(){
        int i,j;
        int mlimits[MAXMT];

        do {

        // Get initial limiting values
        this->printdata();

        for (i=0; i<MAXMT; i++) mlimits[i] = 0;

        for (i=0; i<MAXPT; i++)
                for (j=0; j<this[i].machneeded; j++)
                        mlimits[this[i].ptime[j].type-'A'] += this[i].ptime[j].time * this[i].demand;
```

```
        for (i=0; i<MAXMT; i++) printf("%i,", mlimits[i]); printf("\n");

        // Randomize and modify class ptime parameters.
        for (i=0; i<MAXPT; i++)
                for (j=0; j<this[i].machneeded; j++) {
                        if (mlimits[this[i].ptime[j].type-'A'] <= 0 || this[i].demand == 0)
                                this[i].ptime[j].time = 0;
                        else {
                                this[i].ptime[j].time = rand() % (1 + int(mlimits[this[i].ptime[j].type-
'A'] / this[i].demand));
                                mlimits[this[i].ptime[j].type-'A']  -=  this[i].ptime[j].time  *
this[i].demand;
                                //printf("%i:%i\n",i,this[i].ptime[j].time);
                        }
                }

        scanf("%c",&i);
        } while(1);

}

// ============================================================================
// Randomise processing sequence (verified)

void Product::randsequence(){
        int i,j,r;
        producttimes temp[MAXMT];
        for (i=0; i<MAXPT; i++)
                if(this[i].machneeded > 1) {                    // only run if more than 1
                        for(j=0;j<MAXMT;j++) temp[j].type = ' ';
                        for(j=0;j<this[i].machneeded;j++) {
                                do r = rand() % this[i].machneeded;
                                while(temp[r].type != ' ');
                                temp[r].type = this[i].ptime[j].type;
                                temp[r].time = this[i].ptime[j].time;
                        }
                        for(j=0;j<this[i].machneeded;j++) {// commit changes
                                this[i].ptime[j].type = temp[j].type;
                                this[i].ptime[j].time = temp[j].time;
                        }
                }
}


// ============================================================================
// Print out product data (for debugging)

void Product::printdata(int max) const{
        for(int i=0; i<max; i++) {
                printf("prod %i = %i,",i,this[i].demand);
                for(int j=0; j<this[i].machneeded; j++)
                        printf("%c:%i ",this[i].ptime[j].type,this[i].ptime[j].time);
                printf("\n");
        }
}
```

Buffer.h

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#ifndef BUFFER_H
#define BUFFER_H
#include "tabu.h"

// defining a FIFO buffer
class Buffer {
public:
        void clear_fifo();                      // clear FIFO buffer
        void push_fifo(int,int);    // push on buffer
        int pop_fifo(int);                      // pop from buffer
        int getsize() const { return size; }
        int getprod(int x) const { return prod[x]; }
private:
        int size;                                       // number of items currently in buffer
        int last;                                       // pointing to new location in circular buffer
        int prod[MAXPROD];                      // list of products in buffer
};

#endif
```

Buffer.cpp

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003
// subroutines to implement the FIFO buffer

#include <assert.h>
#include "tabu.h"

// Clearing the fifo buffer
void Buffer::clear_fifo(){
  size = 0;
  last = 0;
  return;
}

// Putting new item into fifo buffer
void Buffer::push_fifo(int max, int item){
  assert(size < max);
  size++;
  prod[last] = item;
  last++;
  if (last >= max) last = 0;
  return;
}

// Removing oldest item from fifo buffer and returning value of that item
int Buffer::pop_fifo(int max){
  int i;
  assert(size > 0);
  i = last - size;
  if ( i < 0 ) i += max;
  size--;
  return prod[i];
```

```
}
```

Cells.h

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#ifndef CELLS_H
#define CELLS_H

#include "fractal.h"
#include "products.h"

// NOT USED YET

/* Cross-header prototypes
struct machine;
struct traffic;
class Location;
class Problems;

*/

#endif
```

Cells.cpp

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#include "fractal.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

// Constructor
Cells::Cells(){
        clearcell();
        totalmach = 0;
        ingress = 0; egress = 0;
        in = NULL; out = NULL;
        for(int i=0; i<MAXMT; i++) machnum[i] = 0;
        cleardemand();
}

// Destructor
Cells::~Cells(){
        if (in != NULL) delete [] in;
        if (out != NULL ) delete [] out;
}

void Cells::operator =(const Cells& source) {
        int i,j;
        pos = source.pos;
        size = source.size;
        totalmach = source.totalmach;
        for (i=0; i<MAXPT; i++) {
```

```
                demand[i] = source.demand[i];
                machnum[i] = source.machnum[i];
                machdem[i] = source.machdem[i];
        }
        for (i=0; i<LIMIT; i++)
                for (j=0; j<LIMIT; j++)
                        m[i][j] = source.m[i][j];
}

void Cells::cleardemand(){
        for(int i=0; i<MAXPT; i++) demand[i] = 0;
        for(i=0; i<MAXMT; i++) machdem[i] = 0;
}

void Cells::resettraf(bool direct){
        int i;
        if (direct == true) // inwards
                for (i=0; i<ingress; i++) in[i].done = false;
        else // outwards
                for (i=0; i<egress; i++) out[i].done = false;
}

void Cells::clearcell(){
        for (int i=0; i<LIMIT; i++)
                for (int j=0; j<LIMIT; j++) {
                        m[i][j].type = ' ';
                        m[i][j].avail = 0;
                }
}

void Cells::setsize(int s){
        calcsq(s,&size);
        assert (size.a <= LIMIT);
        assert (size.b <= LIMIT);
}

void Cells::addmach(int mtype, int amount){
        totalmach += amount;
        setsize(totalmach);
        machnum[mtype] += amount;
        for (int b=0; b<size.b; b++)
                for (int a=0; a<size.a; a++)
                        if(amount > 0 && m[a][b].type == ' ') {
                                m[a][b].type = 'A'+mtype;
                                m[a][b].avail = PERIOD;
                                amount--;
                        }
}

void Cells::rmmach(int mtype, int amount){
        totalmach -= amount;
        machnum[mtype] -= amount;
        setsize(totalmach);
        randomise();
}

int Cells::makepermut(Permutation* pi) const{
        int x = 0;
        for (int b=0; b<size.b; b++)
                for (int a=0; a<size.a; a++) {
```

```
                                pi->perm[x] = m[a][b].type - 'A';
                                x++;
                        }
        return x;
}

void Cells::writepermut(int total,const Permutation* pi, int period){
// Re-new capacity and if Permutation != NULL update machine layout
        int x = 0;
        for (int b=0; b<size.b; b++)
                for (int a=0; a<size.a; a++)
                        if (x < total) {
                                if (pi != NULL) m[a][b].type = 'A' + pi->perm[x];
                                m[a][b].avail = period;
                                x++;
                        }
}

int Cells::findmach(char mtype, int needed, Location* pos) const{
        int a,b, avail = -0xFFFFFFF, dist = -1;
        Location temp;

        // Product has no start location (new incoming or routed in)
        if(pos->a < 0 || pos->a >= size.a || pos->b < 0 || pos->b >= size.b) {
                // searching for machine with highest availability
                for(b=0; b<size.b; b++)
                        for(a=0; a<size.a; a++)
                                if(m[a][b].type == mtype)
                                        if(m[a][b].avail > avail) {
                                                avail = m[a][b].avail;
                                                temp.a = a;
                                                temp.b = b;
                                        }

                //assert (temp.a != -1);
                if (pos->a < 0) dist = temp.a;                  // product entered from the left
                if (pos->a >= size.a) dist = size.a - temp.a -1; // product entered from the right
                if (pos->b < 0) dist = temp.b;                  // product entered from up
                if (pos->b >= size.b) dist = size.b - temp.b -1; // product entered from down
                if (avail < 1) { //needed) {
                        printf("\nCapacity error finding %c with avail. of only %i when %i was
needed\n",mtype,avail,needed);
                        exit (1);
                }

                //pos->a = temp.a; pos->b = temp.b;
                *pos = temp;
                return dist;
        }

        // product has a start Location
        // searching for closest machine that has availability
        int mindist = 0xFFFFFFF;
        //temp.a = pos->a; temp.b = pos->b;
        temp = *pos;
        for(b=0; b<size.b; b++)
                for(a=0; a<size.a; a++)
                        if(m[a][b].type == mtype) {
                                if (m[a][b].avail >= needed) {
                                        dist = abs(pos->a - a) + abs(pos->b - b);
```

```
                                        // option uncommented: force the flow to the right
                                        if (dist < mindist && ( mindist == 0xFFFFFFF || a >=
temp.a )) {
                                                mindist = dist;
                                                temp.a = a;
                                                temp.b = b;
                                        }
                                }
                        }
        //pos->a = temp.a; pos->b = temp.b;
        *pos = temp;
        if (mindist == 0xFFFFFFF) {
                printf("\nCapacity error, cannot find %c with needed avail. of %i\n",mtype,needed);
                exit (1);
        }

        //printf("\n%i finding %c at %i-%i which has avail. of %i",mindist,mtype,pos->a,pos->b,m[pos-
>a][pos->b].avail);
        return (mindist);
}

void Cells::setpos(int currentf, const Location* lim){
        pos.b = int(currentf / lim->a);
        pos.a = currentf % lim->a;
}

void Cells::adddemand(int ptype,int amount,const Product* P) {
        demand[ptype] += amount;
        for (int i=0; i<P[ptype].machneeded; i++)
                machdem[P[ptype].ptime[i].type -'A'] += amount * P[ptype].ptime[i].time;
}

void Cells::rmdemand(int ptype,int amount, const Product* P) {
        demand[ptype] -= amount;
        for (int i=0; i<P[ptype].machneeded; i++)
                machdem[P[ptype].ptime[i].type -'A'] -= amount * P[ptype].ptime[i].time;
}

void Cells::resetmachdem(const Product* P) {
        int i,j;
        for(i=0; i<MAXMT; i++) machdem[i] = 0;
        for (i=0; i<MAXPT; i++)
                for (j=0; j<P[i].machneeded; j++)
                        machdem[P[i].ptime[j].type -'A'] += demand[i] * P[i].ptime[j].time;
}

int Cells::neighsize() const {
        int J = size.a * size.b;
        int x = J * ( J - 1 ) / 2;
        for (int i=0; i<MAXMT; i++)
                if (machnum[i] > 0)
                        x -= machnum[i] * (machnum[i] - 1) / 2;
        x -= (J - totalmach) * (J - totalmach - 1) / 2;
        return x;
}

void Cells::quickdraw(bool disk,FILE *outf) const{
        printf("\n");
        int a,b;
        for (b=0; b<size.b; b++) {
```

```
                printf("\n");
                for (a=0; a<size.a; a++) printf("%c ",m[a][b].type);
        }
        if (disk) {
                for (b=0; b<size.b; b++) {
                        fprintf(outf,"\n");
                        for (a=0; a<size.a; a++) fprintf(outf,"%c ",m[a][b].type);
                }
        }
}

void Cells::randomise(){
        clearcell();
        Location pos;
        int total = totalmach;
        int temp[MAXMT];
        int mach = 0;
        for (int i=0; i<MAXMT; i++) temp[i] = machnum[i];
        while(total > 0) {
                do {
                        pos.a = rand() % size.a;
                        pos.b = rand() % size.b;
                } while (m[pos.a][pos.b].type != ' ');
                m[pos.a][pos.b].avail = PERIOD;
                while(mach < MAXMT && temp[mach] == 0) mach++;
                m[pos.a][pos.b].type = 'A' + mach;
                temp[mach]--;
                total--;
        }
}

int Cells::simulate(const Product* P){
        int i,j,k;
        bool route;
        Location border;
        int tdist = 0;
        int totaldem = 0;
        int tempdem[MAXPT];     // don't want to change real demand
        for (int prod=0; prod<MAXPT; prod++) {
                tempdem[prod] = demand[prod];
                totaldem += demand[prod];
        }
        resettraf(false);
        //for (i=0; i<egress; i++) out[i].done = false;

        Location pos;
        for (i=0; i<totaldem; i++)
                for (prod=0; prod<MAXPT; prod++)
                        if(tempdem[prod]>0) {
                                pos.a = -1; pos.b = 0; // products coming in from the left
                                for (k=0; k<P[prod].machneeded; k++) {
                                        route = false;
                                        for (j=0; j<egress; j++) {
                                                if (out[j].done == false && out[j].pt == prod &&
out[j].mtype == P[prod].ptime[k].type) {
                                                        if (pos.a == -1) pos.a = 0; // routing on first
machine type
                                                        out[j].cmachpos = pos;
                                                        out[j].done = true;
```

```
                                                route = true;
                                                switch (out[j].edge) {
                                                        case left:  tdist += pos.a; pos.a = -
1; break;
                                                        case right: tdist += size.a - pos.a -
1; pos.a = size.a;  break;
                                                        case up:    tdist += pos.b; pos.b =
-1; break;
                                                        case down:  tdist += size.b - pos.b
- 1; pos.b = size.b; break;
                                                }
                                                break;
                                        }
                                }
                                if (route == false) {
                                        tdist                                           +=
findmach(P[prod].ptime[k].type,P[prod].ptime[k].time,&pos);
                                        m[pos.a][pos.b].avail -= P[prod].ptime[k].time;
                                }
                        }
                        tdist += size.a - pos.a - 1; // move out on the right
                        //printf("\n%i leaving cell",size.a - pos.a -1);
                        tempdem[prod]--;
                }

        // incoming products
        for (i=0; i<ingress; i++) {
                switch (in[i].edge) {
                case left: border.a = -1; border.b = 0; break;
                case right: border.a = size.a; border.b = 0; break;
                case up: border.a = 0; border.b = -1; break;
                case down: border.a = 0; border.b = size.b; break;
                }
                tdist += 2 * findmach(in[i].mtype,in[i].pt,&border);   // count twice
                in[i].cmachpos = border;
                m[border.a][border.b].avail -= in[i].pt;
        }
        return tdist;
}

void Cells::addingress(int proctime,char mtype,Location* src){
        if (src->a < pos.a) in[ingress].edge = left;
        if (src->a > pos.a) in[ingress].edge = right;
        if (src->a == pos.a) {
                if (src->b < pos.b) in[ingress].edge = up;
                if (src->b > pos.b) in[ingress].edge = down;
                if (src->b == pos.b) return;
        }
        in[ingress].peer = *src;
        in[ingress].pt = proctime;
        in[ingress].mtype = mtype;
        ingress++;
}

void Cells::addegress(int ptype,char mtype,Location* dst){
        if (dst->a < pos.a) out[egress].edge = left;
        if (dst->a > pos.a) out[egress].edge = right;
        if (dst->a == pos.a) {
                if (dst->b < pos.b) out[egress].edge = up;
                if (dst->b > pos.b) out[egress].edge = down;
```

```
                if (dst->b == pos.b) return;
        }
        out[egress].peer = *dst;
        out[egress].pt = ptype;
        out[egress].mtype = mtype;
        egress++;
}

void Cells::copytraf(bool path,int pos,traffic& copy) const {
        if (path == false) { // outwards
                copy.mtype = out[pos].mtype;
                copy.pt = out[pos].pt;
                copy.edge = out[pos].edge;
                copy.peer = out[pos].peer;
                copy.cmachpos = out[pos].cmachpos;
                copy.done = out[pos].done;
        } else { // inwards
                copy.mtype = in[pos].mtype;
                copy.pt = in[pos].pt;
                copy.edge = in[pos].edge;
                copy.peer = in[pos].peer;
                copy.cmachpos = in[pos].cmachpos;
                copy.done = in[pos].done;
        }
}

void Cells::setdone(bool direct,int pos) {
        if (direct==true)
                in[pos].done = true;
        else
                out[pos].done = true;
}

bool Cells::ability(bool special, int specialmt) const {
// returns whether a cell is able to process all allocated demand
        for (int i=0; i<MAXMT; i++)
                if (machdem[i] > machnum[i] * MU * PERIOD)
                        if ( special == false || i != specialmt )
                                return false;
        return true;
}

int Cells::getexcess(int maxm, int* n, bool special, int specialmt) const {
        int temp, result = -10000;
        double divresult;

        for (int i=0; i<maxm; i++) {
                if ( n == NULL ) {
                        divresult = machdem[i] / double(machnum[i] * MU);
                        temp = int(divresult);
                        if (temp < divresult) temp++;          // round up
                        if ( result < temp ) // (machdem[i] - machnum[i] * MU * PERIOD))
                                if ( special == false || i != specialmt )
                                        result = temp; // int(machdem[i] - machnum[i] * MU *
PERIOD);
                } else {
                        if ( result < (machdem[i] - machnum[i] * MU * PERIOD))
                                if ( special == false || i != specialmt ) {
                                        result = int(machdem[i] - machnum[i] * MU * PERIOD);
                                        *n = machnum[i];
```

```
                                        }
                                }
                        }
                        return result;
}

int Cells::routed(machine* problem, int period) const {
// returns number of problematic machines and returns list of machine types and overcapacity
// just number of routings if problem == NULL
                        int x = 0;
                        for (int i=0; i<MAXMT; i++)
                                if (machdem[i] > machnum[i] * int(MU * period)) {
                                        if (problem != NULL) {
                                                problem[x].type = 'A'+i;
                                                problem[x].avail = machdem[i] - machnum[i] * int(MU * period);
                                        }
                                        x++;
                                }
                        return x;
}

bool Cells::routeprod(int mach, int amount, bool force, int period) {
// if forced increases machdem, otherwise checks whether there is enough capacity first
                        if (force == false)
                                if (machdem[mach] + amount > machnum[mach] * int(MU * period)) return false;
                        machdem[mach] += amount;
                        return true;
}

int Cells::countr(int pos, Problems* rp, const Product* P) const {
                        int j,k,l,m;
                        // need to copy demand numbers
                        int tempdem[MAXPT];
                        for (j=0; j<MAXPT; j++) tempdem[j] = demand[j];
                        int choice, lowestover, routed = 0;
                        int largest, best;
                        int e = rp->excess[pos].avail;
                        int mtype = rp->excess[pos].type;

                        while (e > 0) {
                                routed++;
                                // look for perfect fit first
                                for (j=0; j<MAXPT; j++)
                                        if (tempdem[j] > 0 && e > 0)
                                                for (k=0; k<P[j].machneeded; k++)
                                                        if (P[j].ptime[k].type == mtype)
                                                                if (P[j].ptime[k].time == e ) {
                                                                        e = 0;
                                                                        rp->incprod(pos,j);
                                                                        //printf("%i,",j);
                                                                }
                                // look for 1 product
                                lowestover = 0xFFFFFFF;
                                for (j=0; j<MAXPT; j++)
                                        if (tempdem[j] > 0 && e > 0)
                                                for (k=0; k<P[j].machneeded; k++)
                                                        if (P[j].ptime[k].type == mtype)
                                                                if (P[j].ptime[k].time >= e )
                                                                        if (P[j].ptime[k].time < lowestover) {
                                                                                lowestover = P[j].ptime[k].time;
```

```
                                                choice = j;
                                                e -= lowestover;
                                        }
                if (e < 0) rp->incprod(pos,choice); //printf("%i,",choice);
                // need to route more than 1 product
                if (e > 0) {
                        // look for largest Aj
                        largest = 0;
                        for (j=0; j<MAXPT; j++)
                                if (tempdem[j] > 0)
                                        for (k=0; k<P[j].machneeded; k++)
                                                if (P[j].ptime[k].type == mtype)
                                                        if ( P[j].ptime[k].time > largest ) {
                                                                largest = P[j].ptime[k].time;
                                                                choice = j;
                                                        }
                        // perfect fit lookahead
                        if ( e < 2 * largest ) {
                                best = 0xFFFFFFF;
                                for (j=0; j<MAXPT; j++) {
                                        if (tempdem[j] > 0)
                                                for (k=0; k<P[j].machneeded; k++)
                                                        if (P[j].ptime[k].type == mtype) {
                                                                lowestover = 0xFFFFFFF;
                                                                for (l=0; l<MAXPT; l++)
                                                                        if (tempdem[j] > 0 && e
> 0)
                                                                                for (m=0;
m<P[l].machneeded; m++)
                                                                                        if
(P[l].ptime[m].type == mtype)
        if (P[l].ptime[m].time >= e-P[j].ptime[k].time ) // enough
                if (abs(e - P[j].ptime[k].time - P[l].ptime[m].time) < lowestover)
                        lowestover = abs(e - P[j].ptime[k].time - P[l].ptime[m].time);
                                                }
                                        if (lowestover < best) {
                                                choice = j;
                                                best = lowestover;
                                        }
                                }
                        }
                        tempdem[choice]--;
                        //printf("%i,",choice);
                        rp->incprod(pos,choice);
                        for (k=0; k<P[choice].machneeded; k++)
                                if (P[choice].ptime[k].type == mtype)
                                        e -= P[choice].ptime[k].time;
                }
        }
        rp->excess[pos].avail += abs(e);
        return routed;
}

// === non-member cell functions
void calcsq(int f, Location* lim){
        int x = int(sqrt(double(f)));
        if (x*(x+1) < f) lim->b = x+1;
```

```
        else lim->b = x;
        x = int(float(f)/float(lim->b));
        if (x < (float(f)/float(lim->b))) x++;
        lim->a = x;
}

void getpos(int x, const Location* lim, Location* target) {
        target->b = int(x / lim->a);
        target->a = x % lim->a;
}

int findcell(int f, Cells* c, const Location* pos) {
        Location temp;
        for (int i=0; i<f; i++) {
                c[i].getcellpos(temp);
                if (temp == *pos) return i;
        }
        return -1;
}
```

Problems.h

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#ifndef PROBLEMS_H
#define PROBLEMS_H

#include "fractal.h"

class Problems {
public:
        Problems();
        void clearprobs();
        void incprod(int,int);
        void showprod(int);
        void allocrp(int,int,int,Cells*,const Product*,Routes*,int*,bool,int);
        int probs;          // number of problem machine type
        int routeps[MAXMT];     // how many products need to be routed
        machine excess[MAXMT];                  // how much excess at which machine type
private:
        int prods[MAXMT][MAXPT];      // which are the products?
};

#endif
```

Problems.cpp

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#include "fractal.h"
#include "tabu.h"
#include <stdio.h>
#include <math.h>

Problems::Problems() {
        probs = 0;
```

```
        clearprobs();
}

void Problems::clearprobs() {
        for (int i=0; i<MAXMT; i++) {
                routeps[i] = 0;
                for (int j=0; j<MAXPT; j++)
                        prods[i][j] = 0;
        }
}

void Problems::incprod(int pos, int ptype) {
        prods[pos][ptype]++;
}

void Problems::showprod(int pos) {
        for (int i=0; i<MAXPT; i++)
                if (prods[pos][i] > 0)
                        printf("%i of type %i,",prods[pos][i],i);
        printf("\n");
}

void Problems::allocrp(int pos,int startc,int f,Cells* c,const Product* P,Routes* route,int* total,bool final,int period) {
        Location lim,src,dst;
        calcsq(f,&lim);
        c[startc].getcellpos(src);
        int i,j,k,choice;
        bool fit;
        int* sequence = new int[f];
        rsequence(&src,&lim,f,sequence,c);
        //for (i=0; i<f; i++) printf("%i,",sequence[i]);

        for (i=0; i<MAXPT; i++) {
                while (prods[pos][i] > 0) {
                        j = 1; fit = false;
                        do {
                                for(k=0; k<P[i].machneeded; k++)
                                        if (P[i].ptime[k].type == excess[pos].type) {
                                                fit = c[sequence[j]].routeprod(excess[pos].type-'A',P[i].ptime[k].time,false,period);
                                                if (fit == true) {
                                                        route[startc].amount[sequence[j]]++;
                                                        c[sequence[j]].getcellpos(dst);
                                                        *total += abs(src.a-dst.a) + abs(src.b-dst.b);
                                                        if (final == true) {

        c[startc].addegress(i,excess[pos].type,&dst);

        c[sequence[j]].addingress(P[i].ptime[k].time,excess[pos].type,&src);
                                                        }
                                                }
                                        }
                                j++;
                        } while (fit == false && j < f);
                        if (fit == false) { // product didn't fit anywhere
                                // look for largest capability
                                k = -0x7FFFFFF;
                                for (j=0; j<f; j++)
```

```
                                        if (c[j].getmnum(excess[pos].type-'A') > 0)
                                                if ((c[j].getmnum(excess[pos].type-'A') * int(MU *
period) - c[j].getmachdem(excess[pos].type-'A')) > k) {
                                                        k = c[j].getmnum(excess[pos].type-'A') *
int(MU * period) - c[j].getmachdem(excess[pos].type-'A');
                                                        choice = j;
                                                }
                                // force route to cell with largest capability
                                for(k=0; k<P[i].machneeded; k++)
                                        if (P[i].ptime[k].type == excess[pos].type)
                                                c[choice].routeprod(excess[pos].type-
'A',P[i].ptime[k].time,true,period);
                                route[startc].amount[choice]++;
                                //getpos(choice,&lim,&dst);
                                c[choice].getcellpos(dst);
                                *total += abs(src.a-dst.a) + abs(src.b-dst.b);
                                if (final == true) {
                                        c[startc].addegress(i,excess[pos].type,&dst);
                                        for(k=0; k<P[i].machneeded; k++)
                                                if (P[i].ptime[k].type == excess[pos].type)

        c[choice].addingress(P[i].ptime[k].time,excess[pos].type,&src);
                                }
                        }
                        prods[pos][i]--;
                }
        }
        delete [] sequence;
}

// === non-member ===

void rsequence(Location* src, Location* lim,int f, int*& target, Cells* c) {  // get a sequence of cells to
test for routing
        int i,j;
        Location dst;
        int currentd;
        int* distance = new int[f];
        for (i=0; i<f; i++) {
                c[i].getcellpos(dst);
                distance[i] = abs(src->a-dst.a) + abs(src->b-dst.b);
        }
        currentd = 0; j = 0;
        while(j < f) {
                for (i=0; i<f; i++)
                        if (distance[i] == currentd) {
                                target[j] = i;
                                j++;
                        }
                currentd++;
        }
        delete [] distance;
}

int evalroutes(int f, int maxm, Cells* c, const Product* P, bool final, int period) {
        int total = 0, i, j;
        if (f < 2) return 0;
        Routes* route = new Routes[f];
        Problems* rp = new Problems[f];
//      for (i=0; i<f; i++) {
```

```
//                      c[i].resettraf(true);
//                      c[i].resettraf(false);
//                      c[i].ingress = 0; c[i].egress = 0;
//              }

                for (i=0; i<f; i++) {
                        //c[i].quickdraw();
                        //for(j=0; j<maxp; j++) printf("prod %i has demand of %i\n",j,c[i].getdemand(j));
                        rp[i].probs = c[i].routed(&rp[i].excess[0],period);
                        //printf("%i machine types are problematic\n", rp[i].probs);
                        //for(j=0; j<maxm; j++) printf("Machine type %c has demand of %i\n", 'A'+j,
c[i].getmachdem(j));
                        for(j=0; j<rp[i].probs; j++) {
                                rp[i].routeps[j] = c[i].countr(j,&rp[i],P);
                                //printf("%i products routed to meet type %c excess of
%i\n",rp[i].routeps[j],rp[i].excess[j].type,rp[i].excess[j].avail);
                        }
                }
                for (i=0; i<f; i++)
                        for(j=0; j<rp[i].probs; j++) {
                                c[i].decmachdem(rp[i].excess[j].type - 'A',rp[i].excess[j].avail);
                                rp[i].allocrp(j,i,f,c,P,route,&total,false,period);
                        }
                //for (i=0; i<f; i++)
                //      for(j=0; j<maxm; j++) printf("Machine type %c has demand of %i\n", 'A'+j,
c[i].getmachdem(j));

                if (final == true) {
                        for (i=0; i<f; i++) {
                                for(j=0; j<rp[i].probs; j++)
                                        c[i].egress += rp[i].routeps[j];
                                for(j=0; j<f; j++)
                                        c[i].ingress += route[j].amount[i];
                                c[i].allocingress(); c[i].allocegress();        // Allocate memory
                                c[i].ingress = 0; c[i].egress = 0;
                                //c[i].cleardemand();
                                rp[i].clearprobs(); rp[i].probs = 0;
                        }
                        //assdem(f,P,c);
                        total=0;
                        for (i=0; i<f; i++) c[i].resetmachdem(P);
                        // Set traffic information
                        for (i=0; i<f; i++) {
                                rp[i].probs = c[i].routed(&rp[i].excess[0],period);
                                for(j=0; j<rp[i].probs; j++)
                                        rp[i].routeps[j] = c[i].countr(j,&rp[i],P);
                        }
                        for (i=0; i<f; i++)
                                for(j=0; j<rp[i].probs; j++) {
                                        c[i].decmachdem(rp[i].excess[j].type - 'A',rp[i].excess[j].avail);
                                        rp[i].allocrp(j,i,f,c,P,route,&total,true,period);
                                }
                }
                delete [] rp;
                delete [] route;
//              for (i=0; i<f; i++) printf("\ncell %i: I=%i, E=%i",i,c[i].ingress,c[i].egress);
                return total;
}

int optextpos(int f, int maxm, Cells* c, const Product* P,int period) {
```

```
        int i, solution;
        int neighlen = f * (f-1) / 2;
        Location lim;
        calcsq(f,&lim);
        Permutation pi;
        for (i=0; i<f; i++) pi.perm[i] = i;
        //for (i=0; i<f; i++) c[i].cleardemand();
        // assdem(f, P, c); MUST NOT USE
        for (i=0; i<f; i++) c[i].resetmachdem(P);
        pi.time = evalroutes(f,maxm,c,P,false,period);
        printf("optimising external layout: %i,",pi.time);
        Permutation* neigh = new Permutation[neighlen];
        Permutation best = pi;
        bool allowed;           // internal flag variable
        positions move;         // pair of numbers showing swap move
        Tabulist tabu;          // declare tabu list
        tabu.list_clear();      // empty tabu list

        for (i=0; i<4*f*f; i++) { // moves
                solution = pi.cposneigh(f,maxm,&lim,&neigh[0],c,P,period);

                // === is this solution allowed? ==========================================
                do {
                        move = compare(f,neigh[solution], pi);  // what move was made?
                        allowed = true;                 // assume that move is allowed
                        if (tabu.list_find(move))       // is move tabu?
                                if ( best <= neigh[solution] ) {    // aspiration criterium
                                        neigh[solution].poison();       // make this move unattractive
                                        // find solution again with previous best being poisoned
                                        solution = find_best(solution,neighlen,&neigh[0]);
                                        allowed = false;            // we have to repeat this
                                }
                } while(allowed == false);          // until move is allowed

                if (neigh[solution] < best)  {      // new best found
                        best = neigh[solution];         // set new best
                        printf("%i,",best.time);
                }

                pi = neigh[solution];
                //pi.display(f);
                // === update tabu list ===================================================
                if (tabu.list_size() == TABUCSIZE )
                        tabu.list_pop();        // remove oldest move from tabu list
                tabu.list_push(move);       // insert move into tabu list

        }
        pi = best;
        //for (i=0; i<f; i++) c[i].cleardemand();
        //assdem(f, P, c);
        for (i=0; i<f; i++) c[i].resetmachdem(P);
        for (i = 0; i < f; i++) c[i].setpos(pi.perm[i],&lim);
        pi.time = evalroutes(f,maxm,c,P,true,period);
        delete [] neigh;
        return (pi.time);
}
```

Dtabu.cpp

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, September 2003

#include "fractal.h"
#include "tabu.h"
#include <stdio.h>
#include <stdlib.h>

void dtabu(int f, int total, int maxp, Permutation& pi, const Product* P, Cells* c, compositions comp){
        Permutation best;           // best Permutation found
        Permutation localbest;      // best Permutation in area
        int solution;               // best solution in neighbourhood
        int noimpmoves;                         // number of moves w/o improvement

        // Initialise tabu search
        pi.distribute(f,total,P,c,comp);
        best = pi;
        localbest = pi;

        int neighlen = dneighsize(total,maxp,P);
        //printf("size = %i\n",neighlen);
        Permutation* neigh = new Permutation[neighlen];
        noimpmoves = 0;
        printf("optimising product distribution: ");
        int rmoves = 0;
        do {
                solution = pi.makedneigh(f,total,&neigh[0],c,P,comp);
                solution = find_best(solution,neighlen,&neigh[0]);

                // === is this a local best? ==========================
                if (neigh[solution] < localbest) {    // new local best found
                        localbest = neigh[solution];      // set local best
                        noimpmoves = 0;                   // be patient
                        printf("%i,",localbest.time);
                } else noimpmoves++;                // otherwise lose some patience
                // === is this even the best? ==========================
                if (neigh[solution] < best)  {        // new best found
                        best = neigh[solution];           // set new best
                        //printf("%i,",best.time);
                        //best.display(total);
                }
                // === Did we exceed MAXNOIMP? ==========================
                if (noimpmoves > MAXNOIMP && rmoves <= MAXRMOVES) {
                        //printf("\nrandomizing");
                        rmoves++;
                        noimpmoves = 0;                   // be patient again
                        pi.drand(maxp,P);
                        pi.distribute(f,total,P,c,comp);
                        localbest = pi;                 // reset localbest
                }
                // === MAXNOIMP not exceeded yet, making the move =========
                else {
                        pi = neigh[solution];

                        //printf("%i,",pi.time);
                }
        } while (pi.time > 0 && rmoves <= MAXRMOVES);
        pi = best;
```

```
        delete [] neigh;
}

// ==========================================

int Permutation::makedneigh(int f, int total, Permutation* neigh, Cells* c, const Product* P,compositions
comp){
        // return as soon as better solution is found
  int x = 0;
  int i, j;
  int besttime = 0xFFFFFFF, bestx;
  for ( i = 0; i < total-1; i++) {
          for ( j = i+1; j < total; j++) {
                  if ( perm[i] != perm[j] ) {          // swap made a difference?
      for (int z = 0; z < total; z++)
                          neigh[x].perm[z] = perm[z];      // copy Permutation
                      neigh[x].perm[i] = perm[j];          // swap products
      neigh[x].perm[j] = perm[i];
                      neigh[x].distribute(f,total,P,c,comp);
//                    if (neigh[x].time < time) {
//                            return x;
//                    }
      if (besttime > neigh[x].time) {
        besttime = neigh[x].time;
        bestx = x;
      }
                      x++;
    }
   }
  }
  return bestx;
}

// ==========================================

int Permutation::drand(int maxp, const Product* P){
        // randomise demand Permutation, return length
        int* temp = new int[maxp];
        int total = 0, i,j,x,y,z;
        for (i=0; i<maxp; i++) {
                temp[i] = P[i].demand;
                total += P[i].demand;
        }
        for (i=0; i< total; i++) {
                x = rand() % (total - i);
                y = 1; z = 0;
                for (j=0; j<=x; j++) {
                        while (y > temp[z]) {
                                z++; y = 1;
                        }
                        y++;
                }
                temp[z]--;
                perm[i] = z;
        }
        delete [] temp;
        return total;
}

// ==========================================
```

```
int Permutation::cposneigh(int f,int maxm,Location* lim,Permutation* neigh,Cells* c,const Product*
P,int period) {
        int besttime = 0xFFFFFFF, bestx;
        int i,j,x=0,z;

        for ( i=0; i<f-1; i++) {
                for ( j=i+1; j<f; j++) {
                        for (z = 0; z < f; z++)
                                neigh[x].perm[z] = perm[z];         // copy Permutation
                        neigh[x].perm[i] = perm[j];             // swap products
      neigh[x].perm[j] = perm[i];
                        for (z = 0; z < f; z++) {
                                c[z].setpos(neigh[x].perm[z],lim);
                                //c[z].cleardemand();
                        }
                        //for (z=0; z<f; z++) c[z].cleardemand();
                        //assdem(f,P,c);
                        for (z=0; z<f; z++) c[z].resetmachdem(P);
                        neigh[x].time = evalroutes(f,maxm,c,P,false,period);

                        /*for (z = 0; z < f; z++) {
                                //printf("%i,",neigh[x].perm[z]);
                                c[z].getcellpos(pos);
                                printf("%i: %i,%i, ",z,pos.a,pos.b);
                        }
                        printf("-> %i\n",neigh[x].time); */

                        if (besttime > neigh[x].time) {
        besttime = neigh[x].time;
        bestx = x;
      }
                        x++;
                }
        }
//      for (i=0; i<x; i++) {
//              printf("\n");
//              for (j=0; j<f; j++)
//                      printf("%i,",neigh[i].perm[j]);
//      }

        return bestx;
}

// ============================================

void Permutation::distribute(int f, int total, const Product* P, Cells* c, compositions comp) {
        for (int i=0; i<f; i++) c[i].cleardemand();
        bool flag;
        int specialmt, j = 0;
        if ( comp == special )      // find special machine type
                for (int i=0; i< MAXMT; i++ )
                        if (c[f].getmnum(i) > 0) { specialmt = i; break; }
        for (i = 0; i<total; i++) {
                c[j].adddemand(perm[i],1,P);
                if (comp == special) flag = c[j].ability(true,specialmt);
                else flag = c[j].ability();
                if (flag == false) {
                        c[j].rmdemand(perm[i],1,P);
                        j++; i--; // redo with different cell
```

```
                }
                if(j >= f) { i++; break; }
        }
        time = total - i;
}

// ========================================

int dneighsize(int total,int maxp,const Product* P) {
        int x = total * ( total - 1 ) / 2;
        for (int i=0; i<maxp; i++)
                x -= P[i].demand * (P[i].demand - 1) / 2;
        return x;
}
```

Flexi2.cpp

```
// Program for fractal layout simulation
// Anna M. Lassila, Sheffield Hallam University, March 2004

#include "fractal.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
// similarity to case study is 0 to 100

void copyproddefs(int maxp,const product* P, product* RCase) {        // operator would have been nicer
        int i,j;
        for (i=0; i<MAXPT; i++) {
                RCase[i].number = P[i].number;
                RCase[i].machneeded = P[i].machneeded;
                for (j=0; j<P[i].machneeded; j++) {
                        RCase[i].ptime[j].time = P[i].ptime[j].time;
                        RCase[i].ptime[j].type = P[i].ptime[j].type;
                }
        }
}

void createcase(int sdev,int maxp,const product* P, product* RCase) {
        int i,total=0;
        int chnum, sprod, tprod, chnump;

        bool excess;
        int nummt[MAXMT], testmt[MAXMT];      // machines required of each type

        // reset demand
        for (i=0; i<MAXPT; i++) {
                RCase[i].demand = P[i].demand;
                total += P[i].demand;
        }

        // get limits from original case
        P->getmtdem(maxp,&nummt[0]);

        // how many changes?
        chnum = int(sdev * float(total/100));

        while (chnum > 0) {
                do { sprod = rand() % maxp; } while (RCase[sprod].demand == 0);
```

```
        //      do { tprod = rand() % maxp; } while (tprod == sprod);
                tprod = rand() % maxp;
                if (RCase[sprod].demand > chnum) chnump = 1 + rand() % chnum;
                else chnump = 1 + rand() % RCase[sprod].demand;
                chnum -= chnump;
                RCase[sprod].demand -= chnump;
                RCase[tprod].demand += chnump;
        }

        // make sure we can do the job
        do {
                RCase->getmtdem(maxp,&testmt[0]);
                excess = false;
                for (i=0; i<MAXMT; i++) if (testmt[i] > nummt[i]) excess = true;
                if (excess == true) {
                        do { sprod = rand() % maxp; } while (RCase[sprod].demand == 0);
                        RCase[sprod].demand --;
                }
        } while (excess == true);
}
```

Products.txt

```
# Table containing product sequence and demands
# Use the following format
# Product, demand, 1st machine: time at 1st machine, 2nd machine: time at 2nd machine, etc...
#
1,25,A:7,E:10,B:3
2,15,A:3,C:2,E:11
3,40,E:5,D:2,B:9,A:2,C:1
4,10,B:8,D:6,A:5
5,15,C:3,D:6,B:2,A:4,E:3
6,15,B:3,C:1,E:2,D:5
7,30,D:1,E:3,B:2
8,5,A:8,B:2,E:6,D:3
9,30,E:1,A:2,B:2,C:1,D:3
10,15,C:1,E:5,A:1
```