Enhanced exception handling in sales order processing workflows

Derks, Wouter W.C.. 2019. “Enhanced Exception Handling in Sales Order Processing Workflows”. figshare. https://hdl.handle.net/2134/34298.

Enhanced Exception Handling in Sales Order Processing Workflows

by

Wouter Derks

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the award of

Doctor of Philosophy of Loughborough University

ACKNOWLEDGEMENTS

First of all I am very much obliged to my supervisor Professor Richard H. Weston for his patience, interest, encouragement and support throughout this research study. It has been a privilege to work with him and this research has greatly benefited from his experience.

Further I would like to express my gratefulness to personnel at the industrial partner company Swan Software Ltd., who have provided financial support and necessary information for this study. In particular I would like to thank Mr. Paul Brazier and Ms. Anne Holmes for their kindness.

I am grateful to Loughborough University for providing financial support in the first three years of my study.

Also I would like to thank Dr. Andy West and Dr. Rob Harrison of the Manufacturing Systems Integration Research Institute for providing a part-time research associate post, which helped to continue my research study.

I would like to acknowledge the assistance of many research students, research associates and teachers who have reflected and commented on my work. In particular I would like to thank Dr. Keith Toh, Mr. Ian Coutts, Mr. Alan Hodgson, Mr. David Shorter and Mr. Jack Gascoigne for their support and interest in my work. Specifically I would like to show my gratitude to Mrs. Margaret Carden for her work and coordinating efforts.

Also I would like to thank my family for their support in this period. Finally I am profoundly thankful to my girlfriend Anita and our son Joey for their patience, support and love.

Abstract

In a general study of the literature eleven main areas of research were considered relevant to workflow management and its application in the domain of Sales Order Processing (SOP). It was concluded that exception handling may provide a unifying focus on the dynamic behaviour of workflow systems. Indeed this initial study revealed that current workflow approaches:

1. will overly constrain sales order processors as they seek to handle exceptions by imposing unrealistic and thus rigid ways of working onto their end-users.
2. do not support the needs of workflow system developers very well, since their workflow specifications are time-consuming to develop and maintain and are difficult to verify.

Having made these observations this study conducted a detailed literature study and identified three main classes of workflow approaches, namely: traditional workflow approaches (i.e. SADT/IDEF0, Grai Nets, IDEF3, IEM and CIMOSA), ECA workflow approaches (i.e. WIDE and Rapide) and transactional workflow approaches (i.e. Sagas, ConTracts, and Partial Rollbacks).

Subsequently it was decided to study specific properties of sales order processing exceptions and their effects. Having established the basic nature of a new model of exceptions in sales order processing domains, it was found to be possible to reason about the way activities should be synchronised in the presence of exceptions, and to define a set of requirements for sales order processors. Also workflow system developer requirements were defined.

In the next phase it was concluded that workflow approaches that handle exceptions (i) by executing predefined sequences of compensation activities (such as traditional and ECA workflow approaches do) do support some of the sales order processor requirements needed, but only meet a few of the developer requirements, and (ii) by dynamically generating compensation activities from some base of exception handling knowledge (as implemented by transactional workflow approaches) cannot support many realistic application scenarios, but that relatively their specifications take little to time to develop, are easy to maintain and are simple to check.

A new workflow approach was proposed based on (i) the concept of deploying both standard and exception workflows and (ii) a new definition of workflow 'state' and 'transformation': Whilst a standard workflow captures the state of a workflow based on the assumption that no exception has occurred, exception workflows capture different states of a workflow for exceptions that have occurred. A rule set defines transformations between these workflows. Also a new implementation approach was proposed based on a workflow engine, agent and application components.

Finally in a case study the proposed workflow approach and reviewed workflow approaches were compared with each other. It was observed that the proposed workflow approach can support more realistic application scenarios and meets many of the workflow system developer requirements.

Table of Contents

Table of Figures

Table of Tables

1. Context of research

1.0 Introduction

This research is centred on the innovative application of computational techniques in a domain of concern to many modern manufacturing enterprises, namely the domain of sales order processing. The research seeks to: (1) identify concept improvements that facilitate enhanced specification and enactment of workflows and (2) evaluate aspects of the usefulness of the new concepts, primarily with reference to workflow requirements in sales order processing domains. Consequently the following sub-sections describe the context of this work from 'requirements', 'solutions' and 'outstanding problem' perspectives.

2.0 An abstract view of 'sales order processing requirements'

During recent decades many publications have appeared on sales order processing [Mil58, Sym69, Bur95]. Sales order processing is frequently characterised as an information processing activity that is office-based. In order to fulfil sales orders in effective and efficient ways the various departments involved have to fit in new and modified sales orders into pre-established specific plans in order to meet their objectives (as illustrated in Figure 1). For example (1) the sales department is concerned with activities that should be constrained by the product availability schedule, (2) the despatching department aims to minimise truck deliveries for a given delivery schedule, and so forth. In general it is possible to define the sequence of necessary activities that have to be performed by contributor departments. As many of its steps are repetitive the execution of this process can be optimised over its lifetime.

Although typically hundreds of cases of sales orders will be handled according to established plans, some exceptions will occur while processing sales orders because (A) not enough stock becomes available before the required due date, (B) the customer is not creditworthy, (C) delivery schedules turn out to be costly and (D) a customer wishes to change some properties of an order or to cancel it [Bur95]. In such situations the departments will have to consider changes to established plans and/or changes to the properties of a sales order, usually in an iterative way.

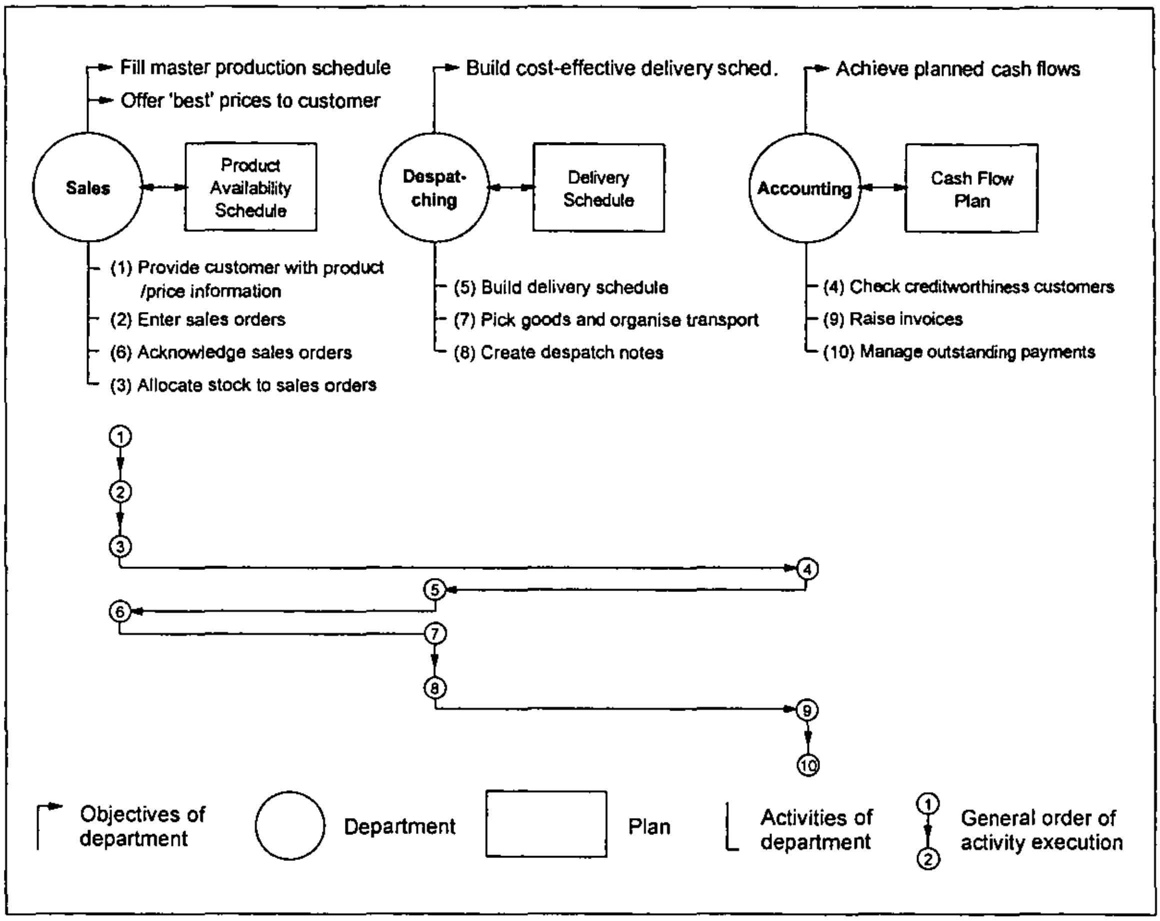

Fill master production schedule
Offer 'best' prices to customer
Build cost-effective delivery sched.
Achieve planned cash flows
Sales
Product Availability Schedule
Despat-ching
Delivery Schedule
Accounting
Cash Flow Plan
(1) Provide customer with product /price information
(2) Enter sales orders
(6) Acknowledge sales orders
(3) Allocate stock to sales orders
(5) Build delivery schedule
(7) Pick goods and organise transport
(8) Create despatch notes
(4) Check creditworthiness customers
(9) Raise invoices
(10) Manage outstanding payments
1
2
3
4
5
6
7
8
9
10
Objectives of department
Department
Plan
Activities of department
General order of activity execution

Figure 1: Characteristics of sales order processing.

Over the last four decades different technologies have been developed and are deployed in support of sales order processing. The following periods can be distinguished and are illustrated in Figure 2:

1. 'Form-based Administration era' [Mil58, Sym69]. The era in which information storage and exchange amongst employees was largely paper-based. As companies became large information was stored in filing cabinets within different departments and routed amongst departments by means of paper forms. Departmental clerks made copies of forms by using carbon paper.
2. 'Islands of Automation era'. In this era stand-alone computers were introduced that stored, processed and retrieved large quantities of information in an efficient and effective way. However during this era information exchange amongst employees remained based on the use of forms.

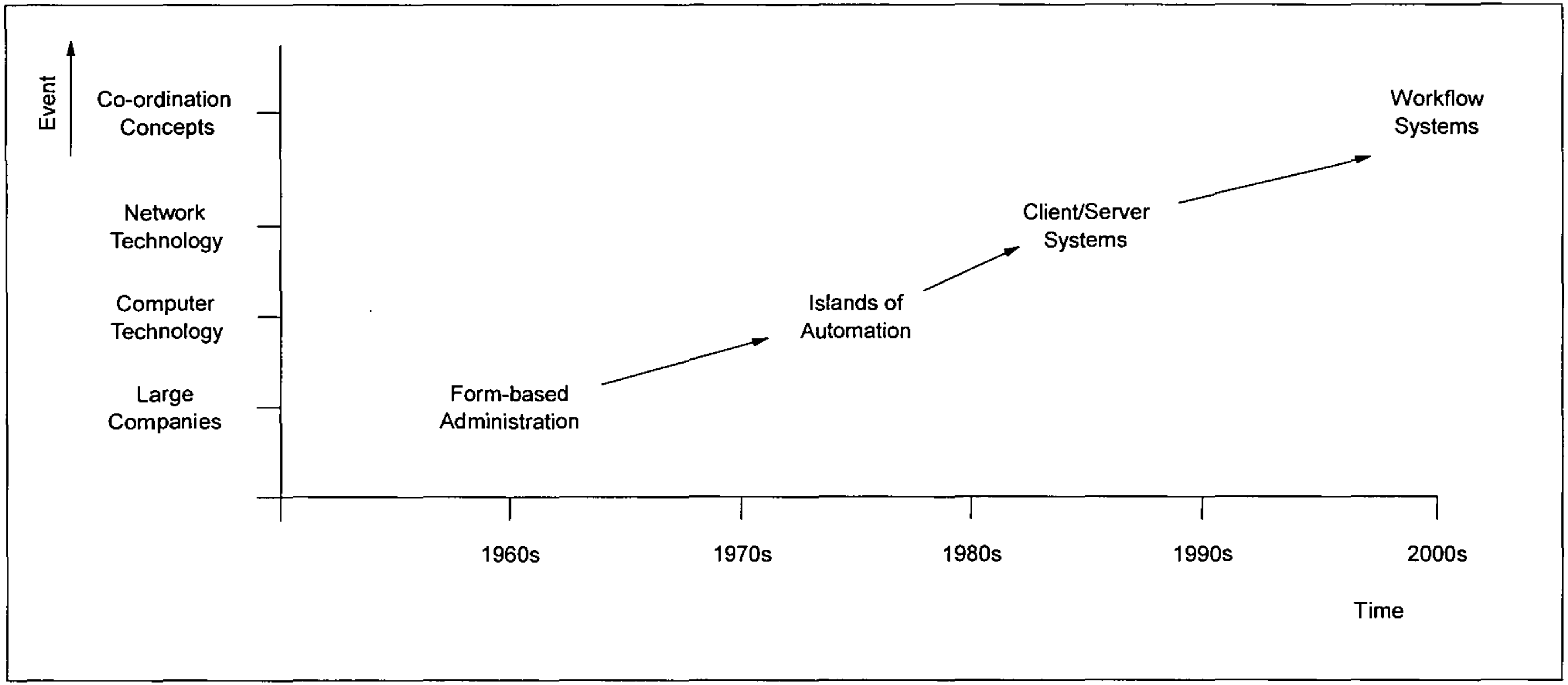


Figure 2: 'Eras' of technology in support of sales order processing.

3. 'Client/Server Systems era' [Orf94, Orf97, Uma97]. In this era network and client/server technology was introduced to facilitate remote access to information stored in a central database.
4. 'Workflow Systems era' [Geo95, Ago00, Sad00b]. The era in which co-ordination technology was introduced to semi-automate certain process steps and to impose co-ordinating structures to ensure correct synchronisation and sequencing of these steps.

3.0 Workflow design & automation and business process engineering methods – 'emerging solutions'

Renewed interest in system development methodologies[1] has been generated following the emergence of business process engineering methods and workflow design and automation, as illustrated by Figure 3 [Im99]. In the last decade a focus of research has been to determine common and necessary characteristics of business process re-engineering (BPR) methodologies and to position BPR methods with respect to other engineering philosophies such as total quality management and continuous improvement [Nei99, Kim96, Ket97, Bra95, Im99, Sim94, Fit96, Vid94, Kal99]. The most frequently quoted definition of business process re-engineering is that of Hammer, namely: "the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed" [Ham93]. Currently accepted BPR tools and techniques include process visualisation, operational method study, change management, benchmarking and process and customer focus [Nei99].

Within the broader context of BPR, workflow systems can play a key role in managing and supporting the work of people who resource business processes. These systems can schedule elements of work, guide users in performing their work, control applications, and track and monitor the progress of work [Chi99, Cug98, Aal00a, Im99]. Workflow systems can function to reduce paper flow times, improve the quality of available enterprise information and reduce communication costs [Geo95].

Business process engineering methods and workflow system approaches are typically interrelated as (A) business process engineering is likely to be a necessary step before developing and implementing a workflow system [Im99], (B) workflow systems may support business process

---

[1] System development methodologies can be broad in scope and encompass multiple disciplines such as the engineering of a company's objectives, strategy, organisation structure, information systems, etc. [AMI93, Bus96, She00, Sim94, Geo95] or alternatively they may be limited to a single discipline such as the engineering of a particular class of information systems (e.g. workflow systems) [Der96, Jab00].

engineering efforts [Bae99, Geo95] and (C) workflow systems are a key candidate technology that can semi-automate the execution of business processes [Ago00, Sad00b, Geo95]. Indeed a common denominator in both business process engineering methods and workflow systems engineering development is process modelling [Gre00, Geo95, Bec00].

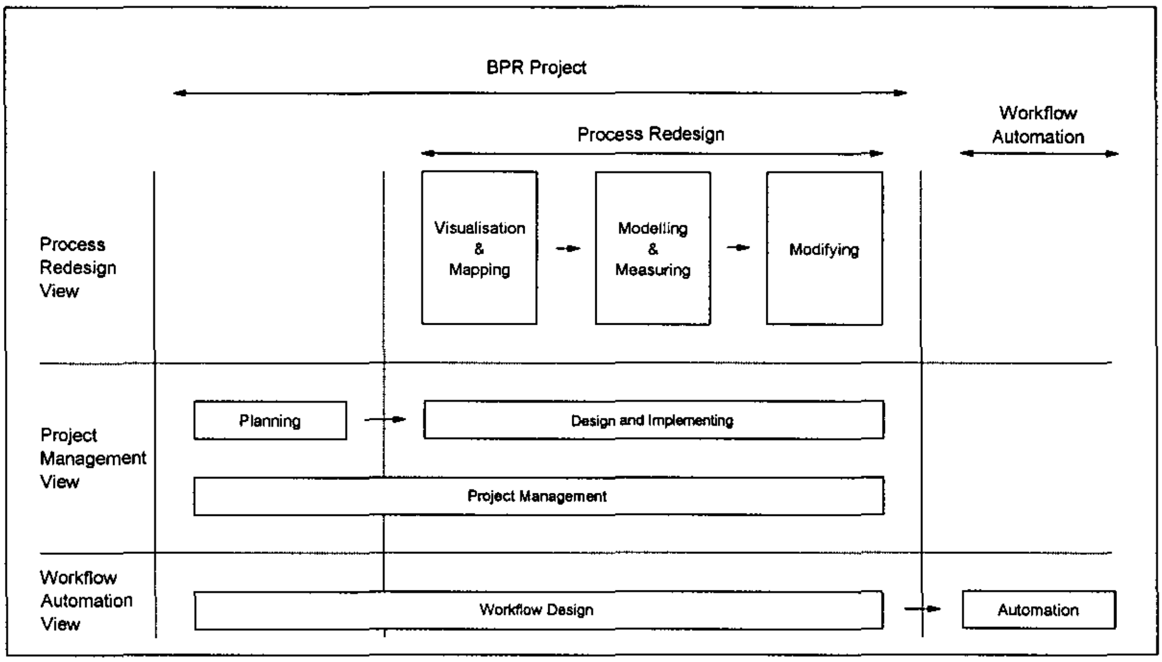


Figure 3: Business process engineering methods and workflow design & automation [Im99].

4.0 Outstanding problems and focus of this study

Over the last two decades computational workflow 'solutions' have advanced significantly from a technological perspective. However the industrial application of this technology has remained constrained for business, social and technical reasons. There is a significant body of recent literature that points out deficiencies of current solutions, but most of this literature presents a technological view. Some of the key observations made are listed in the following.

1. Workflow activity failures cannot be handled readily and effectively in many situations [Alo96, Cug98, Du97, Ede97, Hag00, Kam95, Kam96, She96, She99].
2. The correctness of workflow specifications needs to be improved [Aal98, Sad00a, Aal00b, Aal00c, Aal97, Voo00].
3. Improved workflow simulation is required in support of BPR [Aal01, Hee00, Des00, Bae99].
4. Change capable workflow systems are required [Ago00, Pap99, Gil00].
5. Outstanding organisation and analytical issues remain unsolved [Hol97, Kno98, Zap00, Lee98, Del00]
6. Improved workflow modelling techniques are needed [Jan00, Aal00a, Bec00].

7. Architectural design of workflow systems needs improvement [And00, Bos99, Bar00].
8. Workflow co-ordination and inter-operability issues remain unsolved [Aal99, Kin00, Val98, Dog00, Tid00, Mue00].
9. Participative reasoning about workflow specifications is needed in some application domains [Tra00, Sch96].
10. Workflow change needs itself to be properly defined and enacted as a process [Ell95, Kle00, Del00, Sch98].
11. Various practical problems remain associated with the current generation of workflow systems [Joo97, Kue98, Mey00, Ell00].

A focal point of this study is the specification and development of a new approach to workflow designed to promote an improved use of computational workflow technology in the domain of sales order processing. The present generation of workflow systems is known to be highly capable of supporting routine and repetitive ways of working. However, if workflow execution does not proceed as planned and so called exceptions occur, the use of state-of-the-art workflow technology can severely constrain the way users want to work [Cug98, Cas99, Aal00a]. If unforeseen situations occur because instances of exceptional conditions arise that have not been anticipated and coded up satisfactorily in advance of their occurrence, then these exceptions need to be handled outside the workflow system [Aal00a, Cug98]. Currently state-of-the-art workflow systems are insufficiently flexible to cater for the dynamic nature of business processes [Sad00b]. Hence a prime concern of this research study has been to investigate ways of improving the change capability of new forms of computational workflow technology and to match this capability to general requirements observed in respect of sales order processing domains.

2. Sales order processing characteristics and workflow literature review

1.0 Introduction

This chapter reviews relevant literature about workflows. It is structured primarily with reference to 'workflow requirements', 'workflow solutions' and attendant 'outstanding workflow problems'. Workflow requirements are identified with definitive reference to sales order processing. This has allowed the literature to be analysed in sufficient detail to specify domain requirements that subsequently have shaped a technology specification and provision. Also this application focus has provided a basis for specifying a new approach of workflow systems and for evaluating aspects of the use of this new approach.

2.0 Sales Order Processing – 'domain requirements'

By referring to mainly well established literature about the sales order processing domain this subsection identifies an implicit model of domain requirements.

2.1 Common SOP characteristics found in different company types

This section discusses for a number of company types (A) common features of the nature and the role of sales, (B) important issues that shape sales and (C) some specific and relevant characteristics of sales processes.

2.1.1 Make-To-Stock company type

Make-To-Stock type of companies design and produce products for customers that do not need to be tailored to specific customer needs at their time of purchase. Examples of such standard products include: many types of household foods, furniture, etc.; office equipment and materials; and sub-components of products like nuts and bolts, small-scale integrated circuits, etc. In an MTS type of company, normally the product design is based on generalised customer requirements as anticipated by marketing. Therefore which products are to be produced to stock can commonly be determined by a Master Production Schedule (MPS) which defines the products and quantities that must be manufactured over given periods of time. It is commonplace for MTS-businesses to anticipate production demands by deploying sales forecasting approaches.

It follows that in MTS company types sales order workflows are well-decoupled from make and design workflows. Also important sales processing concerns of MTS companies include: forecasting, order fulfilment and the use of price/discount strategies. The first concern, namely sales forecasting, is concerned with planning demand levels in the long term [Dou95, Olh01, Par02]. Sales forecasting is usually based on statistical methods that may be realised using regression, auto-regressive, moving average, early sales and order over-planning techniques [Bar99b, Kuo99]. More sophisticated forecasting techniques may take causal and event factors into consideration, which may arise as a consequence of promotions, new product introduction and product obsolescence, etc. [Kuo99, Par02]. The second important concern, namely order fulfilment, is concerned with satisfying orders that compete for similar products and particularly where demand cannot be fulfilled for due-dates set by customers [Ho02]. Orders that cannot be fulfilled immediately result in back orders and a loss of sale. Therefore theories have been devised and developed to allocate stock to orders in an optimal way. A third major sales processing concern in MTS environments is the development of price and promotion policies, for example use of a 'while supply lasts' policy [Arc01].

In summary therefore MTS sales processing can be characterised in the following way:

1. A relatively high number of orders and requests that must be handled, typically because of a large customer base.
2. Highly standardised procedures to handle requests for products. Products can be categorised and designated groupings handled in similar ways.
3. Relatively short order and request processing times.
4. A relatively low number of exceptions to the norm when sales processing, as customer needs can only change in limited respects (such as with respect to order due date and quantity).
5. High-level of automation can be deployed to support sales processing.

2.1.2 Assemble-To-Order company type

Assemble-To-Order type of companies design, produce and sell products that can to some extent be tailored to customers' needs, such as by means of invoking so called options and features. A customer can choose from essentially standard products, but from their point of view enhance the product functionality by selecting additional options and features. The design of products, and their commonly needed options and features is based on research activity done by marketing personnel. Numerous examples of ATO products can be found in the transportation, industrial, computer and telecommunications industries [Sha93, Kay96, Dil98]. For example cars can be customized fairly late in their manufacture by enabling the adoption of various engine options, spoilers, wheels, etc. It is also very commonplace for computer systems to be configured (or assembled) via the use of hardware and software options.

Production in ATO companies will normally be based on (1) forecasts of the likely need for standard products, product options and features (rather than being based on a predicted demand for individual end-items) and (2) assembly schedules that are partially modified by customer data encoded within sales orders. In ATO types of business, sales order workflows are less well decoupled from product manufacturing activity than that of MTS types of business. Although the allowable families of product designs and make/assembly plans will need to be determined prior to negotiating sale contracts, the final assembly order is delayed until 'sales contract negotiation' reaches a suitable state.

Sales order processing in Assemble-To-Order type of companies is characterised by four main activity types, namely: order configuration, proposal and quote generation, item allocation and revenue management. The need to configure orders is important as not all combinations of standard products, options and features may be valid [Sha93, Gan95, Tru95, Dil01][2]. Without proper support from product configuration methods and systems, errors can be introduced and the order fulfilment process can become a bottleneck [For02]. Advanced expert-systems and knowledge-based systems may be used to underpin the operation of product configuration systems [Haz86, Sti90, Sha93, Ber94]. Once an order has been configured it becomes possible to generate proposals and quotes [Dil01]. Subsequently there is a need to allocate materials efficiently to customer orders and to replenish exhausted materials. Revenue management is essentially an order acceptance and refusal process that aims to optimise pricing and to re-allocate capacity in order to maximise revenue [Har95].

The sales process can be characterised in the following way:

1. Normally there will be a relatively high number of orders and requests to be handled, but in comparison to MTS businesses the order quantity may be smaller.

2. Standardised procedures will normally be used to handle orders and requests for products. However, (A) product configuration may essentially be a trial-and-error process, as the selection of options or features may exclude other options and features, necessitating restarting of a sub-process and (B) exceptions may require steps to be performed outside the procedure.

3. Longer processing times of ATO sales orders than for MTS sales orders, since in general products will need to be assembled.

4. A greater number of exception types may occur compared to MTS businesses, as changes in customer needs may impact in a greater variety of ways on the configuration of products. Also more exceptions may occur simply because of longer order processing times, during which for example customer needs may change.

---

[2] The implementation of Bills-Of-Material (BOM) for product configuration systems is a research issue in itself. In order to prevent information redundancy when product families are defined, advanced product modelling techniques can be used [Kay96, Ber00].

5. Despite an increase in uncertainty, compared to MTS businesses, ATO companies can still deploy significant automated support for sales processing.

2.1.3 Make-To-Order company type

Make-To-Order type of companies manufacture products that can be customised to a high degree in accordance with needs of customers. After customers have chosen a general kind of product (as researched and specified by marketing personnel), their specific needs will be translated into modifications to the design of the chosen product following some notification of purchase. Generally this means that substantial engineering effort is required which is usually specific to each customer order. An example of make-to-order production is the medium-to-large batch manufacture of specific types of kitchen furniture for large (customer) furniture stores or furniture retailers.

Production in MTOs is normally managed with respect to available and actually needed capacity, i.e. production is not based on some form of forecast but on actual customer needs specified by orders. Therefore in MTO types of business, sales order workflows have to be more closely coupled to design engineering activity flows and product manufacturing activity flows than is the case for ATO types. Typically sales personnel will have to interact closely with design engineering personnel to determine the main options for confirmation with the customer and this will be a driver for subsequent invoicing activity. In practice MTO businesses can vary significantly in respect to the way that sales contract negotiation occurs and therefore in the way that sales activity is able to specialise pre-existing product designs and production plans.

Important issues for sales in Make-To-Order type of companies are lead-time quoting and order selection which serve to maximise the probability of winning orders [Kin93a, Kin93b, Kin96, Ash01]. As products are often highly complex and take some time to produce, it may be appropriate for sales staff to deploy simulation models and/or input/output control procedures to predict/estimate feasible lead times [Kin93a, Cho00]. If estimates and maintenance of lead times are not communicated and co-ordinated between sales and production then orders may be delivered later than promised or loss of revenue can occur. Order selection is important to MTO companies to maximise the total financial gain for the company, as sales and production have different order selection criteria (i.e. concerns about order value/customer desires in contrast to manufacturability) [Cho00, Ash01]. Mathematical models and programming linked to the use of intelligent agents is used to help in the selection of sales orders within some MTO businesses.

Sales processing in MTO companies can be characterised in the following way:

1. Compared with MTS and ATO companies, MTOs are likely to have a relatively small number of orders (but a larger number of potential requests for products) that must be handled.

2. The general procedure followed to process an order in an MTO will be known. Yet many changes to the order configuration may occur; therefore it is likely to be relatively difficult to standardise the flow of sales order processing activities that should be carried out.
3. Relatively long order processing times are likely since products will need to be designed in part and often will need to be produced from scratch.
4. A significantly larger number of types and instances of exception is likely to occur in MTOs, in comparison to sales processes in ATO and MTS businesses. This is because customer needs are normally established on an iterative basis and the expectancy is that they can change over the lifetime of an individual sales order process instance.
5. Only a basic level of automation can currently be used to support sales processing in MTOs.

2.1.4 Engineer-To-Order company type

Engineer-To-Order companies design, make and assemble products in response to specific needs of customers. Often a project will be specified, engineered and produced following a customer request and negotiation process which determines necessary properties of a specialist product or system. Generally this will require complex project engineering so that the concurrent resourcing of parallel threads of activity can lead to timely and efficient: sales contract negotiation and satisfaction, product (or system) design, design for manufacture, product manufacturing, product assembly, system configuration, and product/system test and installation. It follows that in ETO settings sales contract negotiation and satisfaction is normally project specific and although the company may achieve this within a defined framework of procedures, in practice many of the activity flows carried out may occur in an essentially ad hoc manner throughout project life-times. Examples of engineered to order products include the manufacture of complex products such as cars, ships, aeroplanes, etc., or of application domain specific computer systems.

An important issue for sales in Engineer-To-Order companies is bid-preparation [Hir95, Cas97, Cas98]. Frequently sales co-ordinates co-operative activities among specialists from different domains [Hir95]. Also decision-support systems can be used to support sales executives in their determination of an optimum sales price [Cas97, Cas98].

Sales processes in ETO companies can be characterised in the following way:

1. Few orders (but a larger number of requests) that must be handled.
2. Sales processes have many aspects (e.g. legal) and many steps that will need to be taken are not known prior to a given ETO project.
3. Long order processing times are the norm, since products will need to be designed and produced essentially from scratch, but bearing in mind experience and use of previous product and system elements.

4. Many types and instances of exception may occur, because customer needs are normally established in an iterative way. Generally therefore change will be the norm over the timeframe of any given project.
5. Automation has been used to support ETO sales processes, in particular the bidding process is frequently supported.

2.2 General Sales Order Processing characteristics

Many authors have separately explained that the main function of sales order processing is to: (A) provide information on available products and their prices to customers; (B) record orders for products placed by customers; (C) ensure that these products are delivered to the customer at the right place and time and (D) ensure collection of payments for the goods delivered to the customer [Bur95, Mil58, Mur61, Sym69]. Figure 4 illustrates information interchange between sales order processing systems and customers.

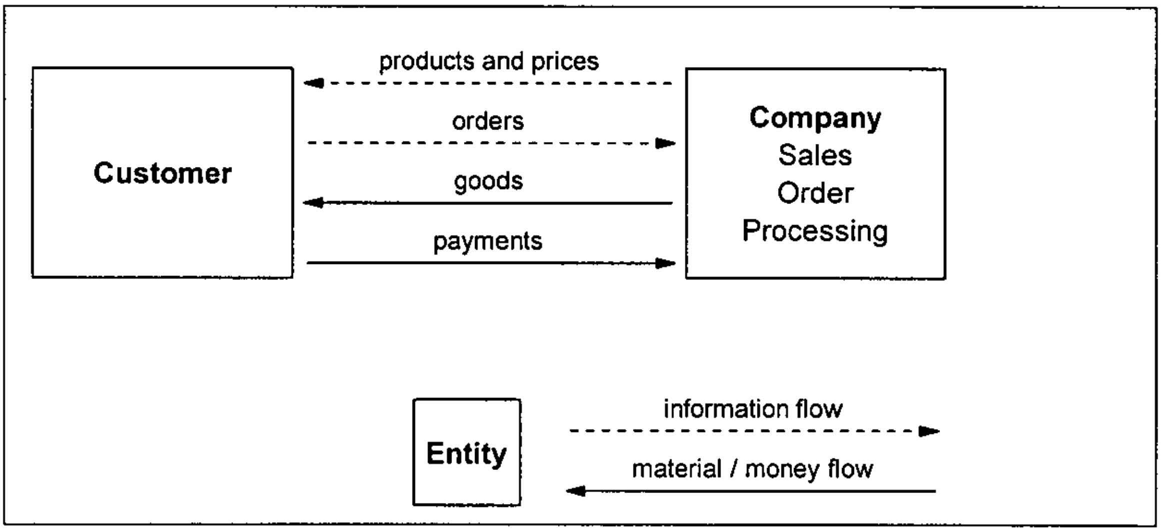


Figure 4: Information, goods and payment flows between customer and company.

Sales order processing systems operate on and/or generate this kind of information so as to facilitate company decision making, e.g. to minimise product holding costs, to minimise the cost of delivering products to customers and to maximise cash flows from the customer to the company. Generally any company deploying a sales order processing system will have defined objectives, will create plans and will formulate policies and rules that enable the planning sales orders in the best way possible whilst taking customer wishes into account [Baa03, Nav03]. Decisions that impact on sales order processing will typically be made in different organisational units, because responsibilities and commitments for sales order processing normally span multiple (and often distributed) organisational boundaries. Examples of organisational units and common objectives, plans, policies and rules of these organisational units are shown in Table 1.

In such schema a 'sales' department will commonly provide customers with information about products and prices, take orders from customers, calculate appropriate prices and discounts, and allocate available stock to customer orders (where this availability is typically recorded in a master production schedule). The objective of the activity 'allocate stock to an order' is to minimise the cost of holding stock. Decisions made during this activity often centre around ways of clustering available stock for given periods. Next sales order information is normally forwarded to 'despatch' and 'accounts' departments.

The 'despatch' department can use sales order information to plan, build and maintain delivery schedules. In so doing they will seek to minimise the cost of product transportation to customers. Common variables associated with this kind of decision making concern customer locations and capabilities of carriers. When a product is due to be delivered appropriate means may be used to pick from stock to deliver them to the customer.

In the 'accounts' department sales orders can be used to update the cash flow plan of specific customers and to send invoices to customers at an appropriate time. The objective of creating cash flow plans is to minimise the risk of customers not paying or paying too late. Decisions are typically based on the level of credit attributed to customers. In return the customer should make a payment for the goods received at an appropriate time.

Organisational unit	Sales	Despatching	Accounting
Objectives	1. Provide product / price information to customers. 2. Fill master production schedules effectively. 3. Determine appropriate prices and discounts.	1. Pick items efficiently (e.g. with respect to some warehouse path). 2. Maintain accurate stock levels. 3. Create cost-effective delivery schedules.	1. Achieve planned cash flows.
Plans	Product availability schedule.	Delivery schedule.	Cash flow plan.
Decision criteria	1. Customer groups or types. 2. Pricing policies. 3. Product availability.	1. Customer locations. 2. Carrier capabilities and costs. 3. Order priority.	1. Customer groups or types. 2. Open payments.
Activities	1. Provide customers with product and price information. 2. Enter sales orders (including configuration data). 3. Calculate prices and discounts. 4. Allocate items to orders. 5. Acknowledge sales orders.	1. Maintain cost-effective delivery schedules. 2. Pick goods and organise transportation. 3. Maintain accurate stock levels. 4. Create dispatch notes.	1. Check customer creditworthiness. 2. Raise invoices. 3. Manage outstanding payments.
Exception examples	1. Customer order change occurs (e.g. quantity, due date). 2. Too little or no stock becomes available to satisfy an order.	1. Too little or no suitable stock available from warehouse. 2. Costly delivery schedule becomes or is found to be inappropriate.	1. Customer is not creditworthy. 2. Goods have been returned. 3. Uncertainty about price to charge.

Table 1: Typical SOP organisational units, objectives, plans, decision criteria, activities and exceptions.

Despite the apparent well-ordered nature of sales order processing activity flows, different types of exception (to the norm) can occur as individual sales orders are processed. Typical exception types are indicated by Table 1. One type of exception occurs should a customer change some of the properties of a sales order after its processing has been initiated. Other departments may have already made decisions based on properties of an original sales order, but as a consequence of the customer change earlier decisions made about product availability, delivery schedules and payment plans may no longer be optimal or even valid. A second type (or cluster of similar specific) exceptions may occur when developing plans and making decisions. For example there may be insufficient products available, with respect to a product availability plan, to meet a customer due date, or it may become evident that a customer is unlikely to satisfy required financial criteria, or it may be discovered that a specified delivery schedule will not be sufficiently cost effective. As a consequence of this second grouping of exceptions it may be decided that it is necessary to change a sales order and to develop alternative plans. A third type (or grouping) of exceptions may occur when seeking to draw actual products from stock, such as where either (a) it is found that they are not actually available or (b) a customer returns or cancels requirements for some products.

3.0 Workflow issues – emerging 'domain solutions' and 'outstanding problems'

This section reviews a number of issues that are emerging in various workflow areas. In many publications on these issues the term exception is used in a rather loose way. Therefore firstly a short review of the terms exception and exception handling is given.

The topic of exception handling is chiefly studied within the context of information system design[3]. It is possible to distinguish between theoretical and applied research into the design of information systems. In the latter case specific aspects of exception detection and handling in application domains (such as design, robot systems, flexible manufacture systems) are investigated. In the former case aspects of programming languages are studied with respect to exception detection and handling.

Fundamental research into the aspects of exception detection and handling have focussed largely on means to detect exceptions, classifying them and handling them for different programming languages, such as functional and object-oriented programming languages [Lis79, Coc82, Knu87,

---

[3] In the area of manufacturing characteristics of exception handling have also been studied in the context of 'management-by-exception' approaches [Mon87, Ric87]. In this context exceptions have been defined as deviations from any kind of plan, e.g. financial budgets. Exceptions can be reported in absolute terms (e.g. dollars) or in percentages.

Phi90, Obe91, Dre94]. Exceptions may be malfunctioning hardware components, design errors, etc. that occur that on irregular occasions [Coc82, Obe91]. Once detected an appropriate exception handler is invoked to resolve the exception where after the normal flow of control may continue. Indeed important aims of research into exception detection and handling are to develop comprehensible program code and reliable information systems [Lis79, Dre94]. Namely, if the program code to handle exceptions is separated from the normal flow of control and stored in special exception handlers, modular information systems can be developed.

Application domains such as design, robot systems, flexible manufacture systems, etc. impose specific exception detection and handling requirements onto information systems. In the domain of design, exceptions occur when design constraints cannot be met [Buc86, Buc88] or when conflicts between design happen [Kle95]. To maintain or to achieve design information consistency it is possible to defer evaluation of constraints (in the former case) or to use conflict management methods (in the latter case). In the domain of robot systems exceptions are errors arising from abnormal operating conditions such as hardware failures, incorrect fixtures and work pieces, etc. [Mas90, Kat93, Mey91, Cox89]. Recovery from such errors (i.e. exception handling) may be performed with the use of expert systems [Mey91]. Thus main aims of information system design in the domain of robot systems are to provide a reliable operating environment as well as to give the robot system a level of autonomy.

3.1 Workflow activity failures

Workflow activity failures are considered to be events where workflow activity execution has not completed successfully (e.g. insufficient stock can be allocated to an order) [Du97]. To handle such exceptions previously performed activities will need to be redone or cancelled. Indeed it is necessary to undo the effects of some previously performed activities that caused or contributed to the failure of the activity [Du97, Hag00, Kam96] so the execution of a workflow instance can continue from a previously valid state [Kam96].

A common approach to implementing exception handling is to embed sequences of compensation activity into a control flow specification [Hag00]. Consider for example Figure 5. The standard workflow for registering a trip of a customer consists of booking a flight (activity A), renting a car (activity B) and booking a hotel (activity E). However some activities may not go as planned. Firstly no flight may be not be available (activity A fails). In this case a train seat is to be booked (activity D) but in case no train seat can be obtained the workflow ends. Secondly if no car can be rented (activity B fails) the flight is of little use and must be cancelled (activity C is performed), but still a train seat can be booked. Thirdly if no room can be booked at hotel 1 (activity E fails), hotel 2 can be tried (activity F). Yet if no hotel at all can be booked all previously performed activities need to be compensated.

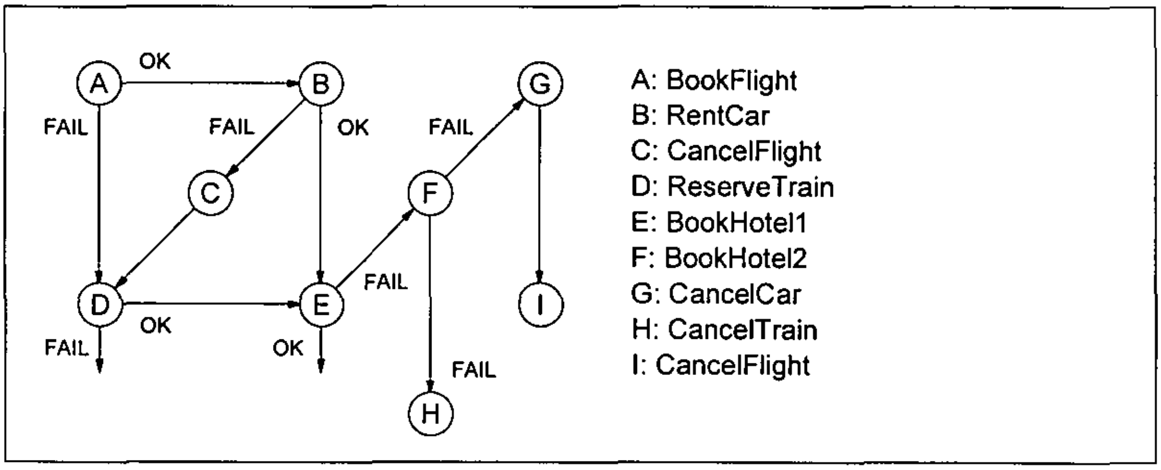


Figure 5: Embedded sequences of compensation activities in a control flow specification.

Two main drawbacks are associated with the embedding of compensation activities in control flow specifications:

1. A workflow designer cannot predict every single case that may occur in the life-span of a workflow [Alo96]. Therefore there will be cases where end-users are not allowed by a workflow system to perform necessary activities. Unless exceptions & their handling has been anticipated and described in the workflow, the system will not tolerate deviations from expected described behaviour [Cug98, Alo97b, She99]. Indeed to handle exceptions users will have to withdraw a workflow from the system (so workflow execution will stop) and perform the required compensation actions manually [Cug98, She96, Alo96]. Indeed it is necessary to go behind the workflow system's back. Once the workflow has been repaired workflow execution is resumed from the point where the failure occurred [Alo96]. However if users are forced to bypass the workflow system frequently, inconsistencies may result between the state of the actual workflow and the state of the workflow as observed by the system [Cug98]. As a consequence workflow systems may become more of a liability than an asset [She99]. Therefore workflow systems have to be flexible enough to capture the requirements of real-world applications [Cug98, Kam96].
2. The interleaving of the original steps with compensation steps makes the control flow specification very complex [Hag00]. By mixing them, the verification and the modification of workflow specifications becomes complicated. As exception handling procedures are based on the order in which the original steps are performed, it becomes difficult to re-use these specifications since they will lack meaning once they are applied outside the context for which they were originally designed [Hag00]. Indeed the current generation of workflow systems lacks an ability to ensure correctness of the workflow execution in the presence of failures [Kam95, Alo97b, Ede97].

3.2 Correctness of workflow specifications

The topic of workflow specification verification is concerned with detecting inconsistencies and errors in large workflow specifications [Aal97, Aal98, Aal00b, Aal00c, Sad00a] and (B) checking similar workflow specifications for equivalence [Voo00]. If mistakes in workflow specifications are first detected at the time of their deployment they are usually corrected in an ad-hoc fashion and at prohibitive cost [Aal00c]. Therefore verification of workflow specifications should (A) point out those parts of a model that contain erroneous or dangerous constructions or (B) indicate those parts of two workflow specifications (based on the same informal description) that are equivalent or are different, and hint at any possible improvements [Voo00]. Verified workflow specifications result in improved throughput times and service levels, and less excess capacity [Sad00a].

Techniques to determine the correctness of workflow specifications centre on identifying two types of structural conflict in control flow models [Sad00a, Aal00b, Aal00c]:

1. Deadlock. A deadlock blocks the continuation of a workflow. A deadlock will occur when two mutually exclusive choice paths are joined by an AND-join (as one of these paths will never be triggered and thus an AND-join will never become enabled).
2. Lack of synchronisation. Lack of synchronisation can result in the multiple activation of a given path in a workflow, when only one activation instance has been intended. It will occur when forked concurrent paths are joined by an OR-join. This is because an instance of the path will be created downstream every time an incoming path completes at an OR-join.

To date two approaches have been developed to detect these two types of structural conflict. One such approach is based on the use of Petri-nets that have a rigorous mathematical foundation and for which a substantial body of theory has been developed [Aal97, Aal98, Aal00b, Aal00c]. Soundness properties of Petri-nets guarantees that (A) any task can be executed by choosing an appropriate route and (B) it is possible to reach a terminal state from any reachable state (thus preventing deadlock). The second approach to detecting structural conflict focuses on the use of graph reduction techniques [Sad00a, Voo00]. Such a reduction process makes use of five reduction rules to remove correct structures from the workflow graph and thereby in an iterative way to identify remaining conflicting uses of modelling structures.

3.3 Workflow simulation in support of BPR

The topic of workflow simulation is concerned with the development of workflow models to analyse and optimise workflows with respect to performance indicators such as flow times, throughput times and resource utilisation [Aal01, Hee00, Des00, Bae99]. Customised workflow specifications increasingly become more important as product and service variety proliferates and their life-time decreases. Yet nowadays few qualitative and quantitative guidelines exist for the design of workflows in the service industry (e.g. banks, insurance companies, governmental departments). By defining workflow aspects such as service time, failure probability, etc. it becomes possible to evaluate different orders in which tasks need to be executed (i.e. combined, in parallel or in sequence) in order to optimise flow times, throughput time and resource utilisation.

Two kinds of performance analysis techniques can be distinguished: qualitative and quantitative analysis [Hee00]. Qualitative analysis is aimed at establishing whether a model meets specific properties. Quantitative analysis is used to calculate the size or level of specific properties. Furthermore quantitative analysis can be categorised into simulation (i.e. approximation) and analytic techniques (i.e. formalisms and mathematical theories such as Markov chains, queuing theory, etc.).

In general a workflow model will consist of a set of tasks, a set of precedence constraints, a set of resource classes, a resource assignment function and a work item arrival function [Aal01, Hee00]. A task has a set-up time, a service time, a failure probability and a reject probability. The set-up time is the time spent on preparations and the service time is the time to process a work item. The failure probability defines the chance that a task will need to be redone. The reject probability refers to the chance that a task does not complete successfully. Precedence constraints define the order in which tasks are executed: either in sequence or in parallel. A resource is a capability that is used to perform a task. The resource assignment function defines how resources are distributed over the various tasks within a workflow. The arrival function indicates the average number of work items that arrive within each time unit. These aspects are modelled using normal, negative-exponential, Poisson or mixed probability distributions. Further some dimensions of a model may be kept fixed in order to focus on other dimensions (e.g. an assumption may be made that sufficient resources are available). Key performance indicators are [Aal01]: (1) Resource utilisation: the extent to which resources are occupied; (2) Maximal throughput: the maximum number of work items that can be completely processed per time unit; (3) Flow time: the necessary time to process a single work item from start to completion.

3.4 Change capable workflow systems

Many authors describe the importance of workflow system architectures and reusable components of workflow systems.

Agostini and De Michelis explain that workflow systems are a prime technology for supporting business process operation [Ago00]. As such they must be flexible so that they can facilitate business process redesign and continuous improvement. They identify necessary capabilities of next generation tools for creating responsive process environments including an ability to: simulate before execution; formally verify key workflow properties; support unambiguous graphical workflow representation; use minimal inputs; support multiple views of the process (through synthesis algorithms and model conversions); automatically derive exception paths with regard to normal acyclic process flows; and automatically enact model change on running instances of workflows whilst protecting them from undesired outcomes.

Papazoglou and Heuvel discuss a methodology to link enterprise models to wrapped legacy system modules that take the form of off-the-shelf (ERP) components [Pap99]. They refer to these models and modules as Business Objects and describe how selected compositions of these objects can be configured and reconfigured to match business process requirements as changing market conditions dictate. The methodology comprises three phases: "enterprise modelling", "reverse engineering" and "meta-model linking". Technologies deployed include CORBA, Enterprise Java Beans and extended and customised versions of UML and IDL.

Bearing in mind the need for workflow systems to support cross-organisational business processes, Gillmann et al state that the literature neglects issues concerned with measuring the performance of system configurations [Gil00]. They propose a synthetic benchmark for order entry workflow systems used in e-commerce application domains. The benchmark parameterises major components of workflow systems with regard to the flexibility of resultant configurations. Their paper underlines the need for configurable systems.

3.5 Organisation and analytical issues

Knolmayer proposed a business rule based technique to decompose business processes in a coherent manner. This technique is based on an extended Event-Condition-Action (ECA) notation. This kind of technique has potential to define modelling guidelines and restrictions on workflow states in a complex organisation [Kno98].

Holt proposed the use of the 'organised activity' (OA) concept [Hol97]. This brings together Petri-net representation with organised ways of planning and motivating repeated and co-ordinated activities with reference to human roles. Holt describes extended Petri-net techniques that provide computer-support for OA as a foundation for business process and workflow modelling.

Zapf and Heinzl present a framework for evaluating generic process design patterns [Zap00]. The framework is based on concepts from organisational theory and operations research and its use facilitates an identification of suitable process design patterns for different application domains.

Lee describes high level Petri-net representations of trade scenarios that can be stored in a "global repository" and downloaded by trading parties as needed for a particular trade [Lee98]. As parties often trade at "arm's length" focus of this work was on developing trustworthy scenarios with sufficient controls and evidentiary documentation.

Dellarocas and Klein define a process element taxonomy as a hierarchy of process element templates [Del00]. They focus on attributing activities (and sub-activities) with properties of the challenges for which they are well suited. The attributes also characterise activities in terms of their failure modes. In a similar way they define resource, goal and assumption taxonomies. The taxonomies have been developed to promote their reuse within an on-line knowledge base that supports the design of robust workflows.

3.6 Workflow modelling techniques

In this subsection properties of emerging computerised workflow technologies are reviewed. This is candidate (solution) technology which can 'structure' and 'enable' activity and interactivity involved in sales order processing domains, so as to provide business benefits.

Janssens et al review eleven fundamentally different techniques proposed to model business processes and related workflows [Jan00]. Collectively these techniques deploy graphical and computer executable modelling constructs that represent: control flows, data flows, support data structures, the beginning and end of procedures, time stamps, aspects of interactions between nets, performance indicators, some aspects of dynamic behaviours and change in workflow systems and aspects of organisational units and resources. Janssens et al compared the eleven techniques with respect to the reuse of workflow models.

Aalst and Hofstede explain how conventional workflow functionality (e.g. task sequencing, split parallelism, join synchronisation and iteration) need to be complemented by descriptions of workflow patterns. Bearing these requirements in mind these authors compared the capabilities of twelve workflow systems [Aal00d].

Becker et al defined guidelines for workflow modelling, simulation and management. Their guidelines are based on a review of six general representational techniques for adjusting models to the perspectives of different types of user and purpose. In complex enterprise environments they emphasise that multiple actors, with different problem and solution perspectives, will be stakeholders in any workflow development [Bec00].

3.7 Architectural design of workflow systems

Anderson and Dyson argue that architecture holds the key to significant software reuse and the corresponding business benefits [And00]. They explain that architecture is about structure rather than function, hence it should meet non-functional requirements of a domain and constrain work that is required to meet functional requirements. They claim therefore that architecture based reuse is more mature than simply carrying out object-oriented analysis followed by component-based design and implementation.

Bosch and Molin explain that the architecture of a software system constrains its quality attributes [Bos99]. Hence architectural design decisions have a major impact on resulting systems. But few methods exist to support the design of software architectures. To address this need Bosch and

Molin propose use of an iterative evaluation and transformation process so that the design of a software architecture satisfies quality needs of a particular domain. Architecture evaluation can be performed via scenarios, simulation, mathematical modelling and reasoning. Architecture transformation can be achieved by imposing an architectural style, imposing an architectural pattern, using a design pattern and converting a quality requirement to functionality.

Barroca et al review software component and architecture developments during the last thirty years [Bar00]. The review concludes that objects are not enough. Rather to achieve distribution, decomposition into a number of collaborating objects is needed. It also concludes that architectures are key and that their architectural design becomes important. But often architecture is an embodiment of a range of applications that can share an architecture but differ in that they deploy specific components. Software and architecture development takes place within the context of an enterprise and one domain architecture may well not satisfy all enterprise requirements.

3.8 Workflow co-ordination and inter-operability

Van der Aalst and Kindler et al propose the use of Petri-net representation techniques, soundness criteria and a composition theorem to allow partner organisations to check the soundness of their own workflows without detailed knowledge about workflows in other organisations [Aal99, Kin00].

Valk proposed the use of so called Business Process Petri nets (BPP-nets) to facilitate model partitioning according to application needs whilst following a process centred approach [Val98]. Appropriately applied BPP-nets can facilitate sub-system modelling with regard to separate objects. This has potential to allow dynamic adaptation when workflow behaviours cross organisational boundaries and change in those behaviours is required.

Dogac et al described an approach to automating and monitoring the flow of control and documents over the Internet amongst different organisations [Dog00]. In this schema, agents handle activities at their site and provide for co-ordination with other system agents by routing electronic documents according to the process description.

Tidhar and Sonenberg describe an approach to specifying distributed systems exhibiting complex behaviours in dynamic environments [Tid00]. Derived from "organisation" and "management theory" they propose a decision making model that can be used at a high level of abstraction during process and organisation design.

Mueller and Rahm proposed a rule-based approach for dynamic adaptation of collaborating workflows to deal with "logical" failures [Mue00]. In their approach workflow collaboration is based on agreements that specify the delivery times and quality of objects a workflow expects from its collaborating partners. By automatically handling logical failures the robustness and correctness of collaborating workflows is improved.

3.9 Participative reasoning about workflow specifications

Trajevski et al assume that the development of complex workflow specifications (1) typically involves experts from different domains and (2) needs to be achieved with *a priori* knowledge of all possible execution scenarios [Tra00]. They propose a "reasoning about actions" technique to formalise the notion of specification correctness. This methodology expresses not only "knowledge" but "ignorance" (unknown semantic values) and the possibility of exceptional situations. Means of testing the consequences of workflow modification are also proposed.

Schäl explained that conventional information centred design does not properly account for different actions taken in reality nor feedback links between actors [Sch96]. They propose a "language-action" perspective on this problem to develop so called "conversation flows". This helps identify "action workflows" and "declarative workflows" and thereby facilitates conversation flow analysis that focused on workflow specification as a set of "co-ordinated activities with a clear objective". The authors claim that the conversation approach can represent reality (i.e. develop a richer picture) better than conventional uses of an information schema approach. The clearer picture can be developed by representing workflow loop closures, relations and responsibilities involved and how a process (and its related procedures and workflows) can be focused on "customer" needs.

3.10 Workflow change as a process

The seminal paper by Ellis et al on "Workflow change is a workflow" explains current generation workflow systems are not capable of facing the ever changing nature of their business environment [Ell95]. They presented an approach to the modelling of so called dynamic change, within workflows, where change to control and data flows is required on the fly, i.e. whilst workflow execution continues.

Klein et al emphasise the importance of systematically handling exceptions through the lifetime of workflows [Kle00]. They propose an exception classification and describe generalised and specific approaches to "preparing for exceptions", "diagnosing exceptions" and "resolving exceptions".

Dellarocas and Klein emphasise the importance of exception handling within the context of developing robust business processes [Del00]. They propose methods for detecting, representing and resolving exceptions in collaborative work processes based on concepts developed within the MIT Process Handbook project.

Scheer explains how ARIS (Architecture of Integrated Information Systems) can bridge the gap between business process modelling and workflow driven applications, from BPR to continuous process improvement [Sch98].

3.11 Practical problems associated with the current generation of workflow systems

Joosten and Schipper explain why it is not atypical for ten modellers to spend over a year mapping and charting a business process and why despite such efforts a workflow project might fail [Joo97]. The authors conclude that often this is not necessarily a fault of the modelling technique but that effective workflow modelling building requires a framework to structure business process innovation.

Kueng explained that although workflow systems have been used for over a decade there are various perceptions about their applicability and utility [Kue98]. Findings from qualitative studies of workflow implementations showed practical benefits in terms of improved group productivity but significant practical application constraints. Meyer made similar observations in a survey of users of SAP/R3 systems [Mey00]. Over 60% of respondents said they perceived potential benefit of using the SAP BWF (Business Workflow) product to interconnect and control distributed applications, namely in terms of reducing throughput times and improving flow control, process reliability, increasing the transparency of information and thence achieving improved quality. But potential drawbacks include high implementation costs and lack of maturity of current workflow products particularly in terms of lack of clear organisational redesign and implementation strategy, minimal consultant expertise and lack of clear and widely accepted concepts for users.

Ellis and Keddara explained that workflow technology will have constrained practical industrial use if workflow applications and systems continue to be designed "from without" [Ell00]. Primarily constraints will arise because current workflow specification and enactment practice does not retain sufficient knowledge about elements of real systems to reason about necessary change when unexpected requirements, conditions and events arise.

3. Scope of research

1.0 Introduction

The focus of this research investigation was developed as a consequence of literature review (outlined in earlier thesis sections) and early investigatory work carried out by the author in collaboration with Swan Software. Swan Software sell ERP (Enterprise Resource Planning) software products and systems primarily for use by SMEs. Although Swan Software do not sell a 'workflow system' product *per se*, some of their ERP system engineering and development work for customers requires complex information flows to be implemented.

This chapter develops a rationale for advancing certain aspects of current computational workflow technology provision so that key application hurdles can be overcome in sales order processing domains.

2.0 Confining the boundaries of research

In section 2.1 of chapter 2 common aspects of sales processes were identified for different types of company. This section explains that the scope of research needed to be limited and why it became focused on a study of (1) sales order processing found in certain company types and (2) selected key aspects of that processing. The following paragraphs detail the study focus more explicitly.

1. Sales order processing found commonly in Make-To-Stock and Assemble-To-Order types of company become the focal point of study. This meant that sales related activity in Make-To-Order and Engineer-To-Order type companies was not considered in detail. In MTS and ATO company types, sales processing can be largely procedural and deterministic in nature, as commonly many product characteristics will have been determined prior to the instantiation of sales processing [Sha93, Dil98, Arc01, Ho02]. Whereas in MTO and ETO companies, sales activity flows[4] are invariably company specific and their organization and resourcing needs to be more fluid [Hir95, Cas97, Cas98].
2. A further important focus is on MTS and ATO operational sales processes, as opposed to tactical and strategic sales processes. Whilst tactical and strategic processes address many issues of relevance to a business in the medium to long term (e.g. planning of master production schedules [Dou95, Olh01, Par02], quality improvement processes), operational processes deal with day-to-day sales issues.

---

[4] In MTO- and ETO-type of companies sales order processing is a term used not so often; instead the term bid-preparation is used more frequently to refer to the process of sales.

3. Another explicit study focus was on data and control flow aspects of sales processing. Previously many different aspects of business processes have been modeled and characterized [AMI93]. However it was determined that this study would not be concerned with issues such as allocating resources to processes, and assigning responsibilities and authorities for process activities to departments, groups, individuals, etc, even though it was understood that such factors can influence specifics of the flow of control in related sales processes.
4. A further specific study focus has been on considering the use of improved automated workflow systems in support of common sales order processing activities in MTS and ATO company types. As the volume of sales transactions in such companies is normally high, and the flow of control is often highly procedural, it is known that automated workflow systems can be used beneficially. However current automated systems are limited in terms of their operational flexibility and robustness. Hence this study focus was chosen with a view to addressing aspects of those constraints.

In section 3.0 of chapter 2 it was observed that the term exception is used with a generalized meaning in many application domains. This study is specifically focused on the nature of exceptions and their handling in certain types of sales order processing domains. The aim was to extend existing knowledge in this domain by identifying and describing new types of exceptions and new ways of handling those exception types. The research study makes no attempt to characterize other kinds of exception found in other application domains, such as exceptions found in robotics or in product design domains[5]. However later in this thesis, namely in chapter 10 section 5.4 contrast is drawn between different sources of exceptions and methods of handling exceptions in 'sales order processing' and this is contrasted against the 'product design and engineering' domain. The purpose here is to begin to generalize findings of this study and to begin to consider their potential reuse in other application domains, such as in support of Concurrent Engineering activities.

Even though it is not an aim of this study to define aspects of exceptions and their handling from a theoretical perspective, one contribution of the study was to provide new insights into the nature and practicality of semi-automated exception handling within MTS and ATO company processes. This study used modeling constructs previously developed by the workflow community, but it did not prove feasible to develop formal theory in support of the use of these constructs. Rather this was left to other researchers active in the area of workflow modeling.

---

[5] Exceptions in robotics have markedly different properties when compared to exceptions in sales order processing, possibly because commonly their origin is related to the control of hardware rather than to theoretical abstractions about workflows.

3.0 Unsolved research requirements with reference to workflows found in sales order processing domains

From section 3 of chapter 2 it is observed that many possible workflow research directions could be explored with reference to unsolved requirements of the sales order processing application domain. Necessarily the scope of this research project has been limited in order to analyse issues in sufficient depth. However it is important to understand certain interrelationships between these research directions so that more generic research issues can be identified. From such a consideration specific focus has been on developing and matching new workflow approaches to common requirements of sales order processing domains.

3.1 Sales order processing workflow failure types

Although many of the general statements made in section 3.1 of chapter 2 about 'workflow activity failure' are found to apply in the sales order processing domain, common activity failures found during sales order processing have yet to be identified and analysed in the literature. Many types of exception may affect a workflow instance in this domain. Logical activity failure is but one class of exception. Some exception types that affect the output of previously performed activities (such as customer order quantity or due date changes) cannot be classified as a logical activity failure. Furthermore the conditions used to reason about exception responses may change during the finite lifetime of a workflow. Consequently decisions may no longer be valid even though exception handling procedures may have already been partly executed. Under such circumstances it may be more costly to handle an exception than to not handle it. Consider for example the case where insufficient stock can be allocated to an order so that an extra order is raised to meet demand. Assume now that another customer cancels an order so that sufficient stock becomes available. Yet an exception handling procedure may have been invoked and cannot be cancelled. Consequently for different reasons it may be necessary to go behind the workflow system's back and to manually make certain changes to a workflow instance. But this raises issues related to consistency. Insufficient rules may be known about how activities are affected by exceptions in sales order processing domains.

Despite these observations, many of the statements made in section 3.1 of chapter 2 about 'workflow activity failures' bear relevance in the sales order processing domain. Indeed more general observations can be drawn from the literature about exceptions in the context of sales order workflows as follows.

Typical examples of sales order processing activities that can fail are 'check creditworthiness customer', 'allocate stock' and 'invoice customer'. Also as a consequence of a change to the output of previously performed workflow activities a sales order workflow instance may be partly or completely invalid. For example in the case of insufficient items being allocated to an order, a customer may want to order 50 items instead of 300 items. In such a case certain properties of a sales order will need to be modified, some stock items will need to be re-allocated and a new order confirmation may need to be sent. But by embedding explicit sequences of compensation activities in control flow specifications, it may become too difficult to modify and verify exception handling procedures when changes to the standard workflow logic occur. Indeed if the standard workflow logic is changed, associated exception handling procedures will also be affected. Consider for example the workflow specification illustrated by Figure 6.

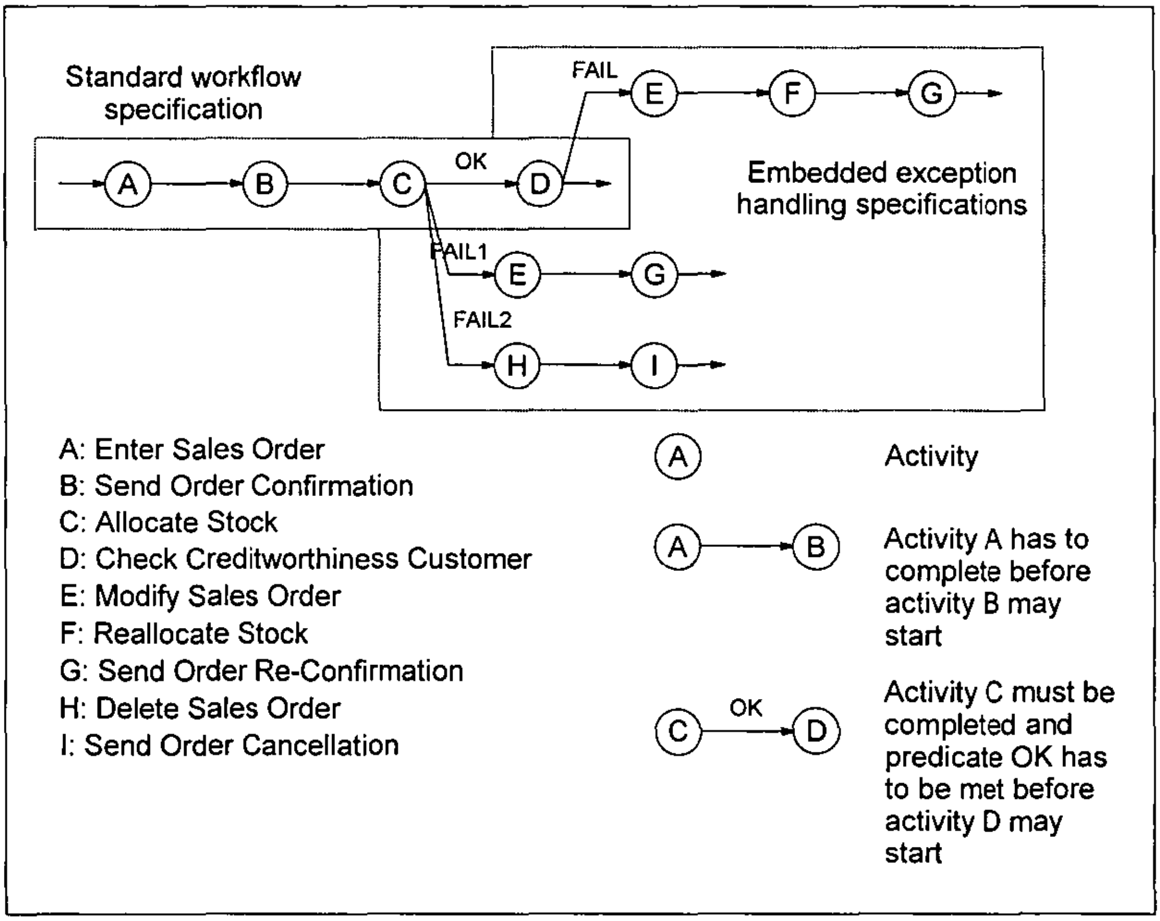

Standard workflow specification
FAIL
E
F
G
A
B
C
OK
D
Embedded exception handling specifications
FAIL1
E
G
FAIL2
H
I
A: Enter Sales Order
B: Send Order Confirmation
C: Allocate Stock
D: Check Creditworthiness Customer
E: Modify Sales Order
F: Reallocate Stock
G: Send Order Re-Confirmation
H: Delete Sales Order
I: Send Order Cancellation
A
Activity
A
B
Activity A has to complete before activity B may start
C
OK
D
Activity C must be completed and predicate OK has to be met before activity D may start

Figure 6: Example exception handling specifications embedded in a workflow specification.

This example standard workflow comprises an ordered sequence of activities A, B, C and D. While a corresponding example of an exception handling procedure is an ordered sequence of activities E, F and G. Now assume that a change occurs to the logic that underpins the standard workflow. Suppose the order confirmation is not sent after the sales order has been entered but that activities 'allocate stock' and 'check creditworthiness' are performed first, so that the new order is A, C, D then B. In such a case all three exception handling scenarios will need to be modified. Because of this kind of phenomenon generally a significant amount of time and effort will be needed to specify and implement changes to exception handling specifications associated with sales order processing workflows. Therefore in practice it is likely that only the most frequently occurring and important types of exception will be analysed and catered for in advance of running a computational workflow system.

3.2 Issues related to the correctness of sales order workflows

To date in the literature, the notion of correctness of workflow specifications has centred largely on detecting deadlocks or on the lack of synchronisation in control flow specifications [Aal97, Aal98, Aal00b, Aal00c, Sad00a, Voo00]. However other notions of correctness may exist with respect to workflow specifications.

Firstly, it is important to determine the correctness of exception handling procedures embedded in workflow specifications. To date few techniques are available to a workflow designer to verify whether an exception handling procedure is appropriate for a given standard workflow and exception instance. As a consequence some SOP exceptions may go unnoticed at the time of design and first appear at the time of execution. For example consider the following standard workflow: (A) 'enter sales order', (B) 'send order confirmation', (C) 'check creditworthiness customer' and (D) 'allocate stock'. Now assume that a condition exists where insufficient stock is allocated to an order (i.e. the activity 'allocate stock' fails). A possible exception handling procedure might consist of compensation activities (K) 'modify sales order' and (L) 're-allocate stock'. Yet this procedure may not be correct as it does not include all needed compensation activities. For example compensating activity 'send order re-confirmation' is missing. It is observed therefore that a consideration of the 'correctness of workflow specifications' should also be focussed on necessary compensation activities and the order in which they must be executed. The notion of correctness may need further extension where certain kinds of exceptions cannot be captured by a workflow approach. To date necessary compensation actions for exceptions such as customer order quantity and due date change cannot be adequately captured using traditional approaches to workflow. Therefore it is important to determine how exceptions and their handling can be captured and how the correctness of representations of exceptions and their method of handling can be verified.

Secondly workflow specifications may encompass other specifications such as a data flow specification, a resource classification, a description of roles, etc. besides control flow specifications. It has been recognised that these specifications must be correct as well [Aal00b, Aal00c]. For example data flow specifications must define the correct flow of data from activity to activity and not leave out any data necessary for processing at an activity. Finally resource classifications, roles, responsibilities, etc must be checked for consistency, yet little work has been done in these areas.

4.0 Research aim

This research study aims to address the requirements of two different groupings of people concerned with the life-cycle of sales order workflows, namely: 'sales order processors' and 'workflow system developers'[6]. Although the roles and concerns of sales order processors and workflow system builders is different, the efficiency and effectiveness with which these kinds of 'actors' can perform their duties is influenced significantly by the 'structure of the workflow approach' deployed[7]. The former class of personnel require a workflow system that: ensures timely processing of sales orders; improves the quality of their own decision making; and ensures that the cost of processing is acceptable. While the latter class of personnel is concerned mainly about the overall effectiveness of business processes for which they are responsible and therefore about the ease and effectiveness with which selected and implemented workflow systems can support the ongoing development and maintenance of actual workflows. Necessarily therefore this study is multidisciplinary in nature as it concerns the development of new understandings about sales order processing domain requirements and alternative ways of utilising leading edge workflow concepts, methods and tools in support of people responsible for the life-cycle engineering of change capable workflows. Both 'technical research' and 'application research' questions needed definition and resolution. Figure 7 illustrates how the focus of this research study straddles two fairly distinctive concerns about workflows and workflow system.

---

[6] Here 'sales order processors' implies those various personnel involved in the operation of a sales process. While use of the term 'workflow system developers' implies various system development roles such as designers and implementers of workflow systems, designers and implementers of specific workflows and personnel concerned with changing specific workflow systems and workflows during their useful life-time.

[7] Here use of the term 'structure of the workflow approach' implies a coverage of structural properties of a workflow system and operational workflows specified and enacted by such a system. Generally these properties will be determined (1) during the conceptual design and realisation of a given workflow system and (2) during specific implementation(s) and uses of that workflow system.

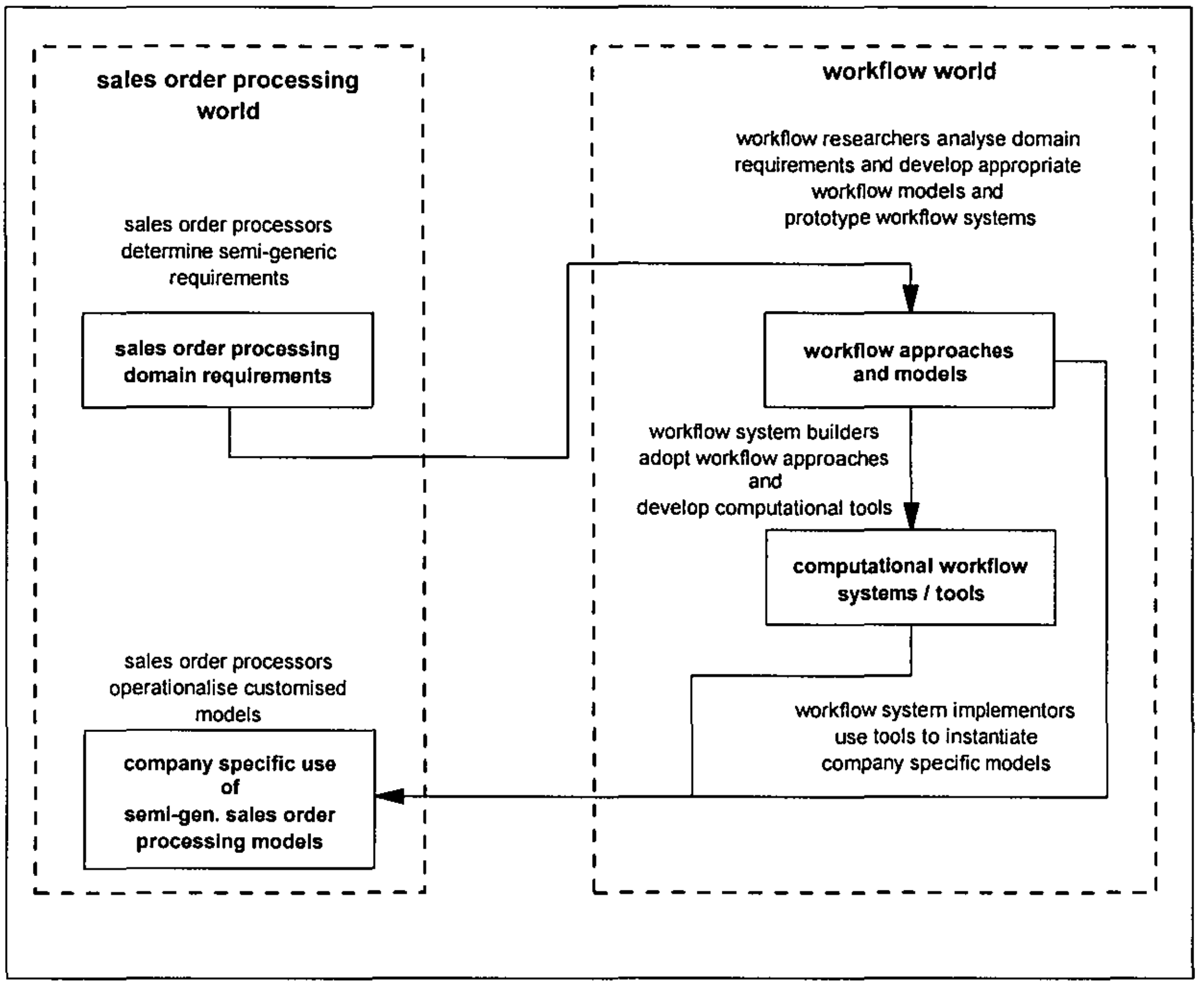

sales order processing world
sales order processors determine semi-generic requirements
sales order processing domain requirements
sales order processors operationalise customised models
company specific use of semi-gen. sales order processing models
workflow world
workflow researchers analyse domain requirements and develop appropriate workflow models and prototype workflow systems
workflow approaches and models
workflow system builders adopt workflow approaches and develop computational tools
computational workflow systems / tools
workflow system implementors use tools to instantiate company specific models

Figure 7: Different concerns and groupings of people involved in establishing workflow approaches and models.

It follows that the notion of change capable sales order processing workflows will be interpreted from various viewpoints that encompass:

1. The sales order processing world (and related roles of various human and technical sales order processors). In this context sales order processing workflows must be capable of reacting to different kind of change that impact on the state of a workflow, as defined by the results of activities that have been performed by a sales order workflow at a specific instance in time. Typical conditions that give rise to change (i.e. exceptions), and are considered in this study, include 'customer order change', 'stock allocation problems' and 'customer creditworthiness problems'. Consequently stock may need re-allocation, delivery schedules may need to be amended, an order re-confirmations to be send, etc. Thus interaction between sales order processor activities will typically be required. Measures used to evaluate the impact of this kind of change include 'order throughput times', 'processing costs', 'consistency of plans' and 'available support for decision making'.

2. The workflow world (and related roles of researchers, system implementers and system builders). In this context change in sales order processing workflows will be considered primarily to result from changes made to workflow specifications (i.e. the defined order and conditions under which activities should be executed) that are made in response to changing business and operational requirements. The ease with which exception handling specifications can be adapted, depends largely on the system's approach to the definition of workflow state and transformation rules. Measures used to quantify the ability of a workflow system to respond to changing business and operational requirements will centre on the time required to develop and enact new workflow specifications as well as the time required to modify and check workflow specifications for their correctness.

Two key, interrelated research ideas will be pursued in this study whilst developing a new 'workflow approach' and implementing and evaluating instances of that approach. The two ideas are as follows:

1. It may prove beneficial to separate flows of work into 'standard' and 'exception' workflows rather than consider workflow instances to comprise a single workflow. This first idea is illustrated by Figure 8. It is proposed that standard workflows will capture relatively static, established and commonly used relationships linking normal work activities but that exception workflows will more effectively capture relatively transient and conditional interdependencies between activities that must be satisfied in the event of instances of change (to the standard workflow). When all changes specified by an exception workflow are accepted it is assumed that their necessary modifications will be mapped onto a standard workflow so the standard workflow 'jumps' to a new valid state from which sales order processing can continue. Furthermore it is assumed that it will be necessary to update or cancel some parts of exception workflows so that they can be deployed for alternative change scenarios as and when exception conditions occur. Also it is assumed to be necessary to evaluate the value of multiple exception workflows. By such means it is proposed that proper use of exception workflows could result in change-capable workflow systems and change capable workflow instances.

2. That for a given state of a standard workflow an appropriate exception workflow can be generated dynamically by using some form of knowledge about the class of exception that has occurred and its related general data and control flow dependencies. Rather than attempt to pre-specify explicit sequences of activities at the time of workflow design, so as to cover every possible exception and workflow state. The idea here is to develop a set of rules that are specific to the sales order processing application domain, that can be used to predict and dynamically generate necessary compensation activities at workflow execution time. But it is presumed that exception workflows can only be generated for exception types that have had appropriate consideration at the time of workflow design and/or specification. In this way a

new approach to handling exceptions is to be defined so the developed approach will facilitate change capability in sales order processing workflows.

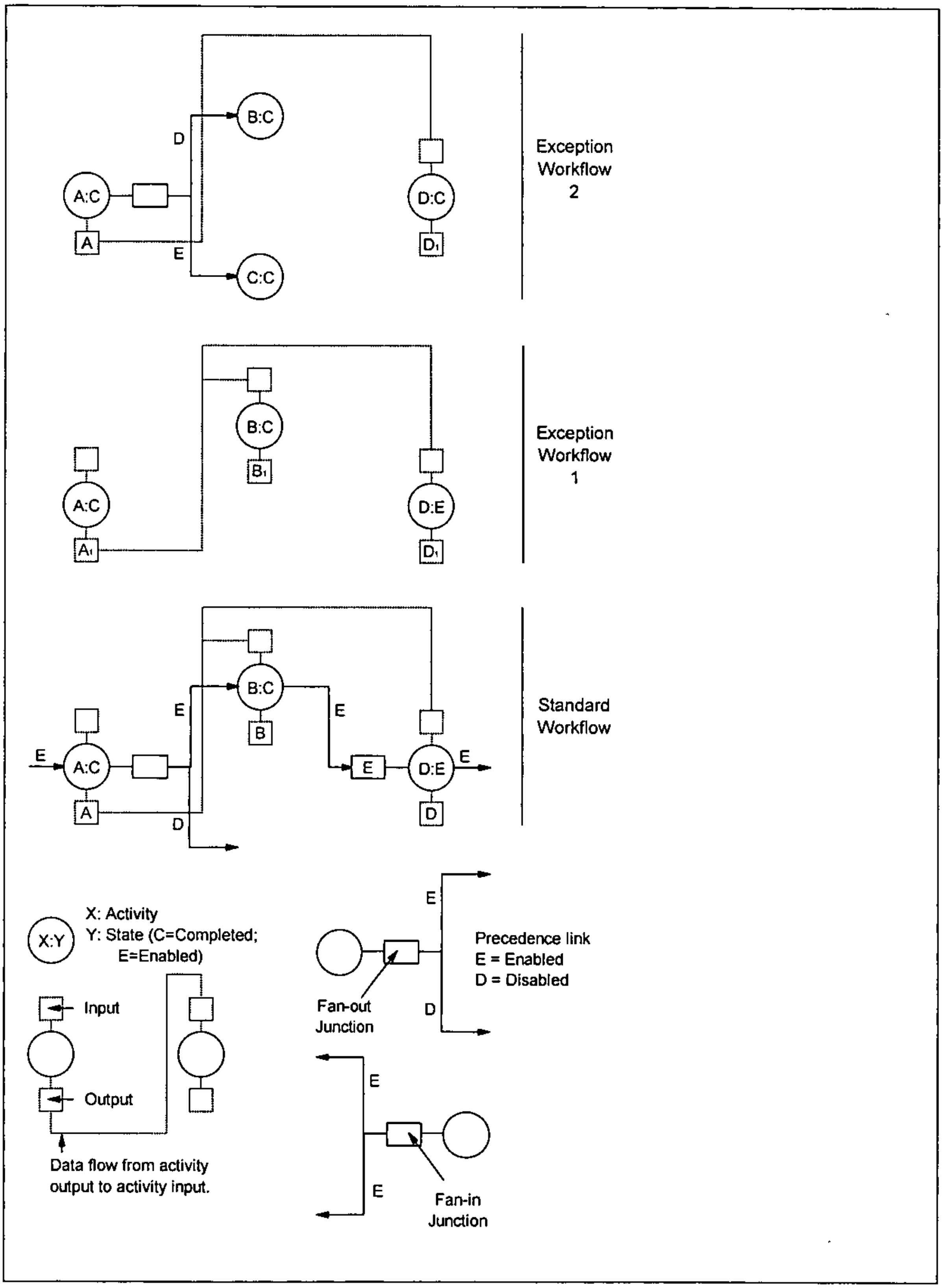


Figure 8: Standard and exception workflows.

By pursuing research investigation into ideas 1 and 2 it is hoped that improved support will be given to:

(I) Sales order processors, because (A) for all possible exceptions and workflow states it may prove effective to generate an appropriate exception handling procedure and (B) it should prove possible to reverse decisions made at an earlier stage with respect to handling exceptions, so as to cancel or update parts of workflows where necessary.

(II) Workflow system developers, because (A) it is no longer necessary to predetermine the impact and pre-program responses needed for every possible exception instance and workflow state, and (B) fewer exception handling procedures will require review and change when cases of business rule change occur. Here it is presumed that new decompositions designed into workflow systems and workflow instances will help workflow system developers to cope better with growth in workflow complexity. The validity of this presumption is also to be partially evaluated in this study. If it proves to be true, potentially at least this could significantly impact on the industrial practicability and utility of next generation workflow systems.

5.0 Workflow system prototyping and testing

It was determined that an evaluation environment would be specified and developed to part test the capabilities and applicability of the concepts incorporated into the new generic and domain specific workflow approaches. It was decided that as far as proved practical and effective implementation of the new approach would conform to the guidelines for workflow systems specified in the reference model of the Workflow Management Coalition (WfMC) [WFM94]. The WfMC model consists of a set of general software components and a set of general interfaces (an abstract view of this model is illustrated by Figure 9). The main component classes of a WfMC system are (A) business process analysis modelling & definitions tools and (B) workflow enactment services. An aim of the WfMC model is to standardise workflow components and interfaces so as to promote the generation of software products from different vendors and to facilitate their interoperation. The WfMC claim that its workflow standards for inter-working are designed to (A) allow application dependent choice of "best of breed" products within any specific workflow implementation (so as to deliver multi-vendor workflow solutions) and (B) ongoing support for operational and business re-engineering.

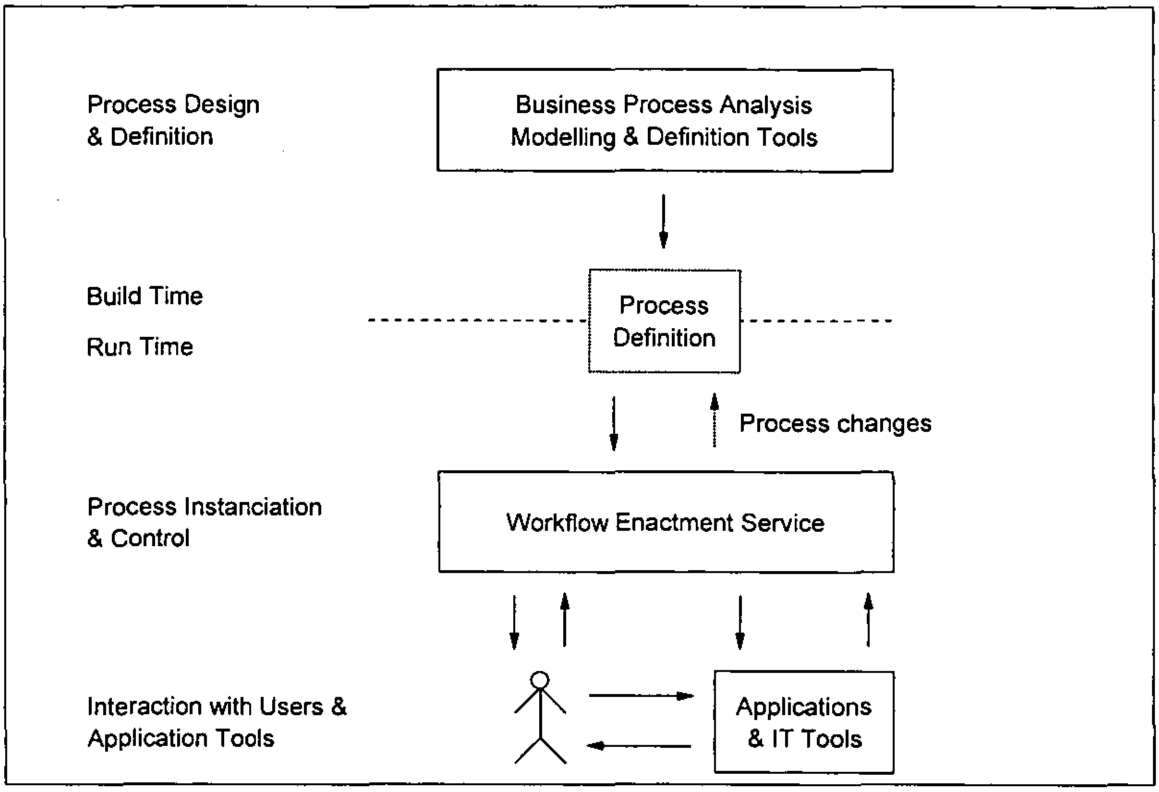


Figure 9: Reference model of the Workflow Management Coalition [WFM94].

The purpose of the WfMC business process analysis modelling & definition tools is to (A) create formal definitions of relevant aspects of workflows, (B) perform workflow simulations, and (C) analyse the correctness of workflow definitions. Specifications of workflows will usually include computer representations of activities, their starting and completion conditions, rules governing the execution of activities, and associated roles and other organisational structure aspects. The purpose of the workflow enactment service is to create and control actual workflow instances as they need to occur in the real operational environment. This entails (A) scheduling various activities within workflows, (B) transferring data and control between activities, and (C) invoking applications that perform activities. Furthermore logical workflow definitions may be partitioned into different workflow software components (e.g. the workflow enactment service may consist of multiple co-operating workflow engines each managing part of the overall execution of a workflow definition).

6.0 Research objectives

To achieve the research aims outlined in section 4.0 of this chapter a suitable research methodology was determined. The methodology adopted included the following phases of research activity (as illustrated by Figure 10).

6.1 Detailed literature review

A detailed literature review of current workflow concepts was carried out to determine candidate approaches and mechanisms for handling workflow exceptions. The main approaches reviewed included 'traditional', 'event-condition-action' and 'transactional' means. The results of this detailed review are reported in chapter 4. Elementary techniques integrated into these workflow engineering approaches were identified and characterised so as to provide an understanding of state-of-the-art techniques that might be reused when developing new, more robust workflow systems in terms of their capability to cope with exceptions.

6.2 Study and modelling of sales order processing exceptions and their impact

A focus of this set of research activities was an identification and specification of a 'logical model of sales order processing activities and ways in which exceptions impact on those activities'. The main purposes of this logical model was seen to be: (A) to characterise and classify exception types and define their properties that arise in sales order processing, such as customer order change, customer creditworthiness problems, etc., (B) to characterise the effects of exceptions on previously performed activities and on activities to be performed next and (C) to help identify alternative ways of synchronising activity execution. It was also envisaged that this logical model of sales order processing would facilitate the design, implementation and evaluation of a new and robust approach to workflow specification and enactment. Chapter 5 describes how logical modelling of exceptions in sales order processing domains was carried out and explains how new domain requirements were represented for use in new and robust workflow systems.

6.3 Domain analysis in support of determining means of handling exceptions

Research activity referred to under section 6.1 provided a detailed understanding of contemporary workflow approaches and their general capabilities to handle exceptions. Further sales order processing domain analysis allowed: (A) observation and comparison of specific advantages and disadvantages of state-of-the-art workflow approaches in the light of the ability to satisfy common sales order processing requirements (i.e. completeness) and (B) inherent capabilities of these approaches to be assessed in terms of effort, flexibility and correctness of workflow specifications. The domain analysis and observations made are reported in chapter 6.

6.4 Design and part-implementation of a new approach to robust workflow specification and change

The prime purpose of research activities under sections 6.1 and 6.3 respectively was to inform the conception, specification and development of a novel approach to workflow specification and change that is capable of dealing with common classes of exceptions that arise in sales order processing domains. This set of research activities were carried out to design and part implement such an approach by, where possible, building on the use of best available techniques embodied in state-of-the-art approaches (as defined in chapter 4). Chapter 7 reports on the development of this novel approach.

6.5 Benchmark new and contemporary workflow approaches

The aim of this last set of research activities was to compare identified workflow approaches in a more quantitative way. In chapter 6 general advantages and disadvantages of contemporary workflow approaches were characterised in terms of their completeness, flexibility, and correctness measures. But it was determined that these measures also need to be expressed using measures of quality, time and cost. Hence chapter 9 reports on research activity designed to benchmark advantages and disadvantages of the new workflow approach against previous approaches with reference to characteristic properties of workflows found in an Assemble-To-Order (ATO) type of company. Hence a main focus of the evaluation activities was to (1) identify general properties of typical sales order workflows with reference to general characteristics of ATO businesses, (2) identify, classify and characterise various exceptions that can occur, (3) define the potential impact of those exceptions on the outcome of previously performed activities and (4) appraise the capabilities of the new and previously existing workflow approaches to compensate for those exceptions.

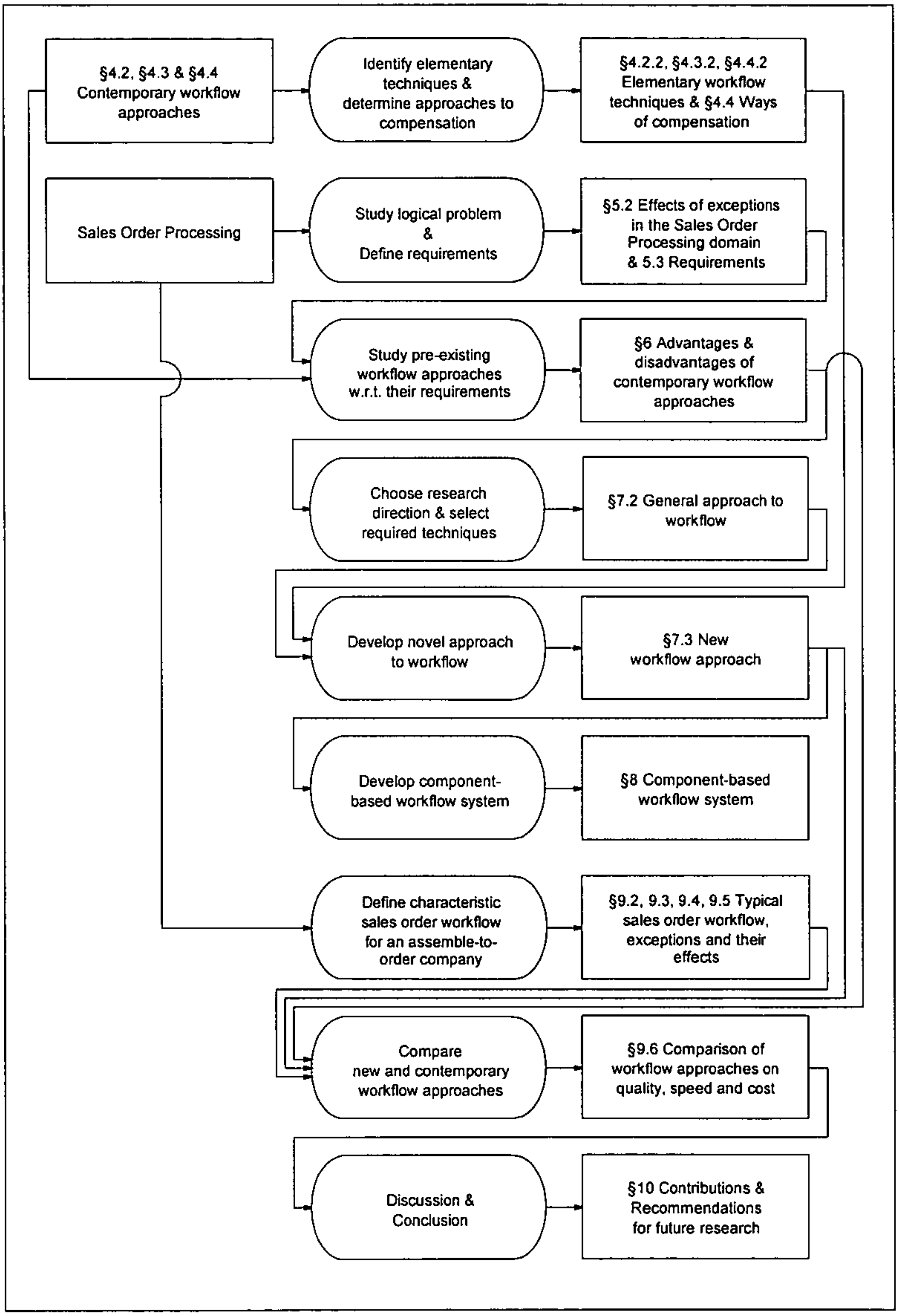

§4.2, §4.3 & §4.4 Contemporary workflow approaches
Identify elementary techniques & determine approaches to compensation
§4.2.2, §4.3.2, §4.4.2 Elementary workflow techniques & §4.4 Ways of compensation
Sales Order Processing
Study logical problem & Define requirements
§5.2 Effects of exceptions in the Sales Order Processing domain & 5.3 Requirements
Study pre-existing workflow approaches w.r.t. their requirements
§6 Advantages & disadvantages of contemporary workflow approaches
Choose research direction & select required techniques
§7.2 General approach to workflow
Develop novel approach to workflow
§7.3 New workflow approach
Develop component-based workflow system
§8 Component-based workflow system
Define characteristic sales order workflow for an assemble-to-order company
§9.2, 9.3, 9.4, 9.5 Typical sales order workflow, exceptions and their effects
Compare new and contemporary workflow approaches
§9.6 Comparison of workflow approaches on quality, speed and cost
Discussion & Conclusion
§10 Contributions & Recommendations for future research

Figure 10: Research plan, primary activities and deliverables.

4. Detailed survey of candidate workflow approaches

1.0 Introduction

This chapter will review concepts and techniques from a broad set of modelling and enactment approaches that may be of relevance to handling exceptions. Many of these selected and reviewed approaches have not been developed with a view to managing workflows, yet are capable of modelling and/or enacting orders in which activities must be executed. Indeed research into workflow approaches is a relatively new area that may draw from widely established knowledge. This study will focus on three different kinds of approach to modelling and enacting systems, namely:

- Enterprise engineering methods [Ros77a, Ros77b, ICA81, Pun84, Mar88, May92, AMI93, Mer95, Spu96, Ver96]. Enterprise engineering methods were developed primarily for the manufacturing community during the period late 1980s to early 1990s. Many such developments were linked to emerging needs for business process modelling.
- Active database systems engineering [Luc95a, Luc95b, Cas96, Wid96, Bar96]. Active database systems research was a particularly popular and important area of research during the late 1980s and early 1990s.
- Advanced transaction systems engineering approaches [Bre93, Alo97b, She96, Alo97a, Du97, Kam98]. Advanced transaction approaches were developed by the database and transaction community in the late 1990s. Their development was linked mainly to emerging needs of non-traditional database applications such as in office automation, CAD/CAM and software engineering application domains.

Whilst enterprise engineering methods are primarily aimed at modelling and specifying relevant aspects of systems, active database systems and advanced transactional approaches centre on ways of enacting systems.

This study analysed potential capabilities of the three approaches and in so doing it was necessary to use a consistent set of terms and measures of performance. This allowed their capabilities to be positioned and compared. For example, it was observed that enterprise engineering methods (such as CIM-OSA) emphasise the importance of different phases in the life-cycle of models and with respect to these phases typically provide a modelling framework and modelling constructs to represent activities, data, resources and so on. In fact the sub sets of concepts and techniques associated with enterprise engineering methods are referred to in the remainder of this chapter as 'traditional workflow approaches' because early in the development of workflow technology they were used to specify various aspects of workflow systems.

2.0 Traditional workflow approaches

Within the context of this study traditional workflow approaches[8] are considered to be approaches that model the order of activity execution by means of data flows and control flows. Examples of workflow approaches that belong to this class are IDEF3[9] [May92], CIM-OSA[10] [AMI93, Ver96], IEM[11] [Mer95, Spu96], SADT/IDEF0 [Ros77a, Ros77b, ICA81, Mar88] and Grai Nets [Pun84]. In the late 1970s and early 1980s data flow issues were considered to be of prime concern when modelling relationships between activities. In this respect the literature reports the development of the data flow modelling methods SADT/IDEF0 and Grai Nets[12]. In the early 1990s came the development of enhanced approaches such as IDEF3, CIM-OSA and IEM that represent activity execution by means of precedence links and fan-in/out junction constructs. Although these second-generation modelling approaches also incorporate techniques to model the flow of data (for example the use of IDEF3 can be combined with that of IDEF0), relationships between flow of data and flow of control are not well modelled. Rather these second-generation modelling

---

[8] Characteristically this class of workflow approach is based upon the use of process modelling in order to specify aspects of workflows, following which transformed versions of these models are enacted in a computer executable form. This has been termed a 'traditional workflow approach' because such an approach has been applied in many current office-based workflow systems [Geo95]. However it was also noted that specific applications may only develop and deploy limited modelling views. Typically contemporary office-based applications provide mechanisms that enable control of the flow of work (and have been termed workflow systems by their vendors) but do not fully order activity execution, such as by specifying and enacting formal descriptions of precedence links and fan-in/out constructs.

[9] Whereas IDEF3 has been specifically developed to model the order of activity execution, CIM-OSA and IEM are much broader in scope having been developed for general enterprise engineering purposes. However if only the behavioural aspects of CIM-OSA and IEM are taken into account they have very similar characteristics to IDEF3. Therefore when thesis references are made to CIM-OSA and IEM approach, essentially reference is made to behavioural features of these approaches unless specifically indicated otherwise.

[10] Essentially CIM-OSA is an enterprise engineering methodology that supports the development of general enterprise models. These models can be particularised for use in specific industries and thereby facilitate management of the life-cycle of company specific models [AMI93].

[11] IEM places enterprise engineering within a context of object-oriented thinking. Thereby it focuses on the re-use of data elements (i.e. orders, products, and resources) and on the re-use of specified orders of activity execution [Mer95].

[12] Even though SADT/IDEF0 is intended as a communication tool and abstracts away from detailed decision making and timing issues, it has also been acknowledged that (1) it can order the execution of activities (at its lowest modelling levels) and (2) can place "constraints" on the flow of data and on the order in which activities are executed [ICA81].

approaches adopt a largely process-centred approach to synchronising the execution of activities. Currently these second-generation models are considered to represent state-of-the-art modelling techniques by the manufacturing community [Ver96].

2.1 Review traditional workflow approaches

2.1.1 IDEF0

The Integrated DEFinition language 0 (IDEF0) is a modelling method developed to capture data requirements of a system from a process perspective [ICA81]. IDEF0 identifies those functions that a system performs and what is needed to perform those functions. The modelling method was developed within the Air Force's Integrated Computer Aided Manufacturing (ICAM) program and is based on the well established Structured Analysis and Design Technique (SADT) [Ros77a, Ros77b, Mar88]. The use of IDEF0 modelling notation has become widespread in business, industry and government world wide.

Primary modelling constructs of IDEF0 are 'function' (represented on diagrams by boxes), and 'data and objects' that define interfaces between functions (represented by arrows connecting boxes). A function can be an activity, a process or a transformation that occurs over time and has recognisable results. It is defined by a verb, or a verb phrase, that describes what must be accomplished. Four kinds of data and object flows are defined that each have a specific role to play (as illustrated by Figure 11):

1. Inputs. Inputs are transformed or consumed by a function and thereby produce outputs. Arrows representing input data and object flows enter the left side of a box.
2. Controls. Controls are the conditions required for a function to produce correct outputs. Controls constrain the execution of a function. Arrows representing control data and object flows enter the top side of a box.
3. Outputs. Outputs are produced by a function that operates on its inputs. Arrows representing output data and object flows leave the right side of a box.
4. Mechanisms. Mechanisms are the means (e.g. people, machines, systems) used to support the function execution. Arrows representing mechanism data and object flows enter the bottom side of a box.

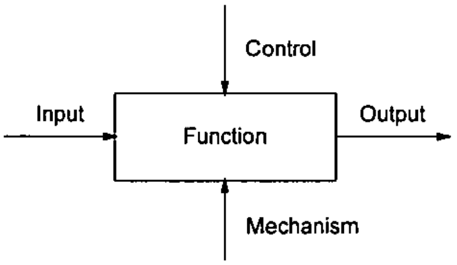


Figure 11: Four principal flows of data and objects.

Thus the so called Input-Control-Output-Mechanism (ICOM) arrows that flow into and out of a function represent all the data and objects that are needed for a function to achieve its purpose. ICOM flows also place "constraints" on the execution of a function since the function may use no other data. Further a function may generate different outputs under different circumstances as defined by its inputs and controls. These different performances result from so called different activations of a function. Both inputs and controls may be transformed by a function into outputs. Also output flows from one function may be connected as input, control or mechanism flows to other functions. Further data and objects may flow from one function to a number of other functions and one function may use data and objects from a number of other functions. Data and object flows are modelled by arrow segments, labels and fork and join junctions (as illustrated by Figure 12).

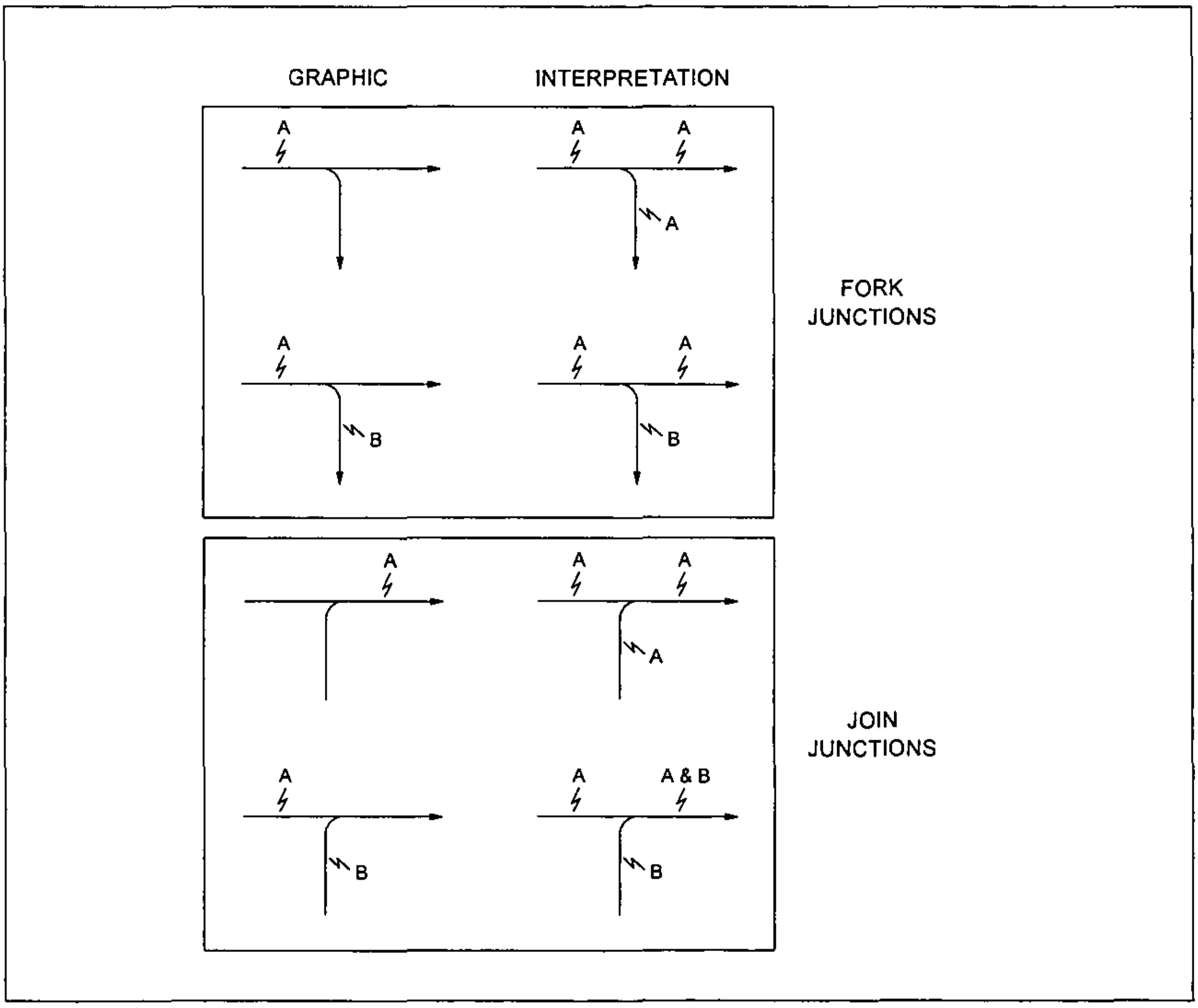


Figure 12: Bundling and unbundling of arrow segment meaning.

A fork junction is a junction type at which an arrow segment divides into two or more arrow segments (connecting a source to more than one destination). This denotes the unbundling of arrow segment meaning that has been combined under a general label. A join junction is a junction type at which a number of arrow segments merge into a single arrow segment (connecting more than one source to a single destination). This denotes the bundling of separate arrow segment meanings into a more general meaning. Arrow segments are labelled with nouns or noun phrases to describe their meaning.

An IDEF0 model is defined by a set of diagrams that decompose or compose functions in a hierarchical manner (as illustrated by Figure 13).

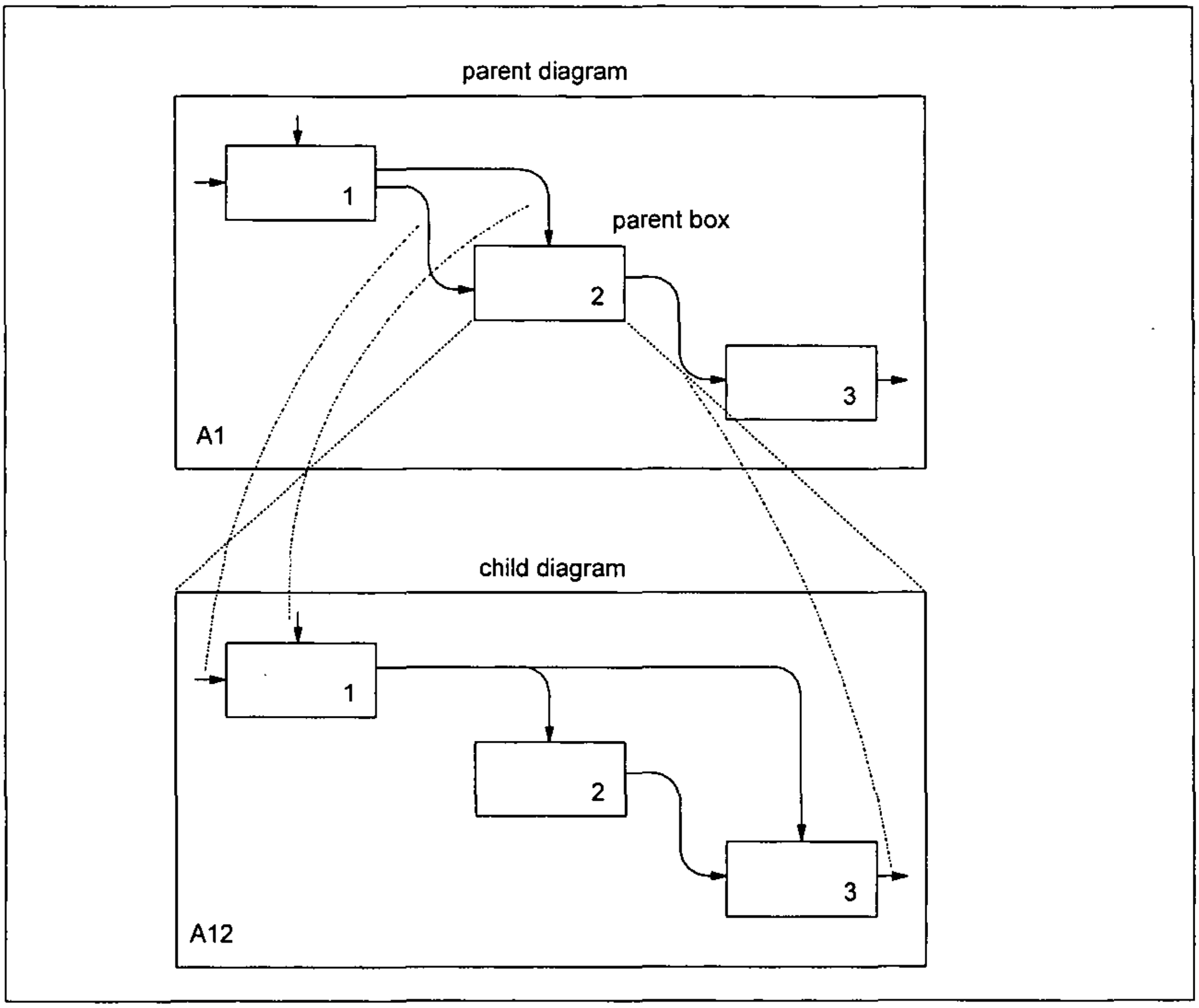


Figure 13: Hierarchical (de-)composition of functions.

The top level diagram is used to represent the main function or functions. Such a top level diagram is referred to as an A-0 diagram. Child diagrams represent a set of sub functions that collectively compose a function described by a parent diagram. The IDEF0 modelling method dictates that any function represented in a parent diagram can be decomposed into at least 3 but at most 6 sub functions in a child diagram. Thus functions can be decomposed into sub-functions, sub-functions into other (smaller grained) sub-sub-functions, and so on. It follows that top level functions are more general and abstract in their definition whereas low level definitions of functions can be very detailed and concrete. In this way the IDEF0 modelling method can gradually introduce increased detail. Each arrow that is connected to a box representing a super function (i.e. a parent box) is considered to be a boundary arrow and shall appear on the child diagram of that function and must enter or leave one of the sub functions on the child diagram (i.e.

a child diagram must have the same scope as the parent box it details). In the top level diagram interfaces between the (main) function or functions and other functions outside the modelled system (i.e. the context) establish the scope of the model. The top level diagram is accompanied by brief statements that specify the model's viewpoint and purpose. Not every function modelled by an IDEF0 diagram needs to be decomposed. An arrow on a parent box may have a different role in a child diagram (e.g. a control arrow on a parent box may be either an input or a control arrow for boxes represented by its child diagram).

2.1.2 Integrated Enterprise Modelling

Integrated Enterprise Modelling (IEM) adopts an object-oriented approach to modelling [Mer95, Spu96]. The philosophy of object-orientation 'enables an integration of function and data aspects'. IEM can be used to provide both an implementation independent logical description of a workflow and a dependent implementation description.

The logical model of such a workflow comprises an information model and a function model that are combined by means of a so-called Generic Activity Model. This is illustrated by Figure 14. The information model is composed of a set of objects that are derived from three predefined IEM-objects, namely: 'Product', 'Order' and 'Resource'. The object 'Product' represents and stores product characteristics such as geometry, material, quality, user functions and production properties. The object 'Order' describes all information needed to plan and control enterprise activities (e.g. quantities required, due-date, places, authorisation, planning horizon). The object 'Resource' describes all material and informational resources needed to execute enterprise activities (e.g. tools, fixtures, equipment for data processing, people, material, information). Technically an IEM object comprises data objects. IEM data objects consist of data items and/or data objects. By using predefined IEM objects it becomes possible to define common data objects that are useful within an industrial domain or in a particular company.

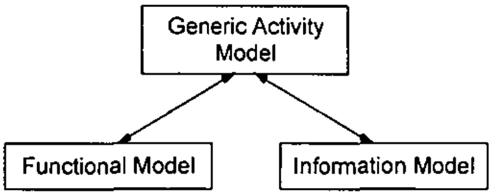


Figure 14: Logical workflow model.

The IEM function model consists of a set of activities and constructs that connect activities, as illustrated by Figure 15. Key properties of the IEM function model are listed as follows:

1. Sequential: activities can be executed in a defined sequence.
2. Parallel: activities can be executed simultaneously and hence shifted in time.
3. Case distinction: only one activity is executed from some choice of activities depending on certain conditions.

4. Loop: an activity is executed repeatedly until some predefined condition is met.
5. Unification: an activity is performed after preceding activities have been completed.

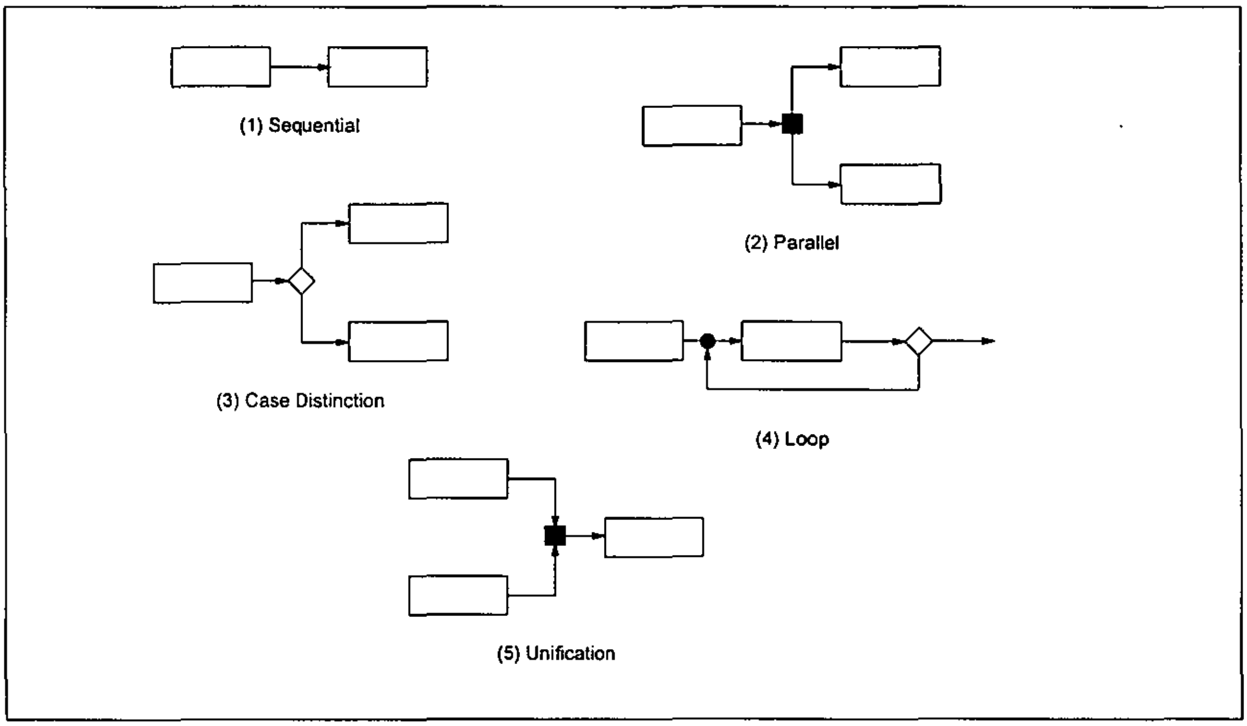


Figure 15: IEM modelling constructs used to connect activities.

The Generic Activity Model links the information model with the function model by means of a description of a set of activities. An activity is defined by the state (n) of its 'Product', 'Order' and 'Resource' input objects prior to its execution, and the state (n+1) of its 'Product', 'Order' and 'Resource' output objects subsequent to its execution, as illustrated by Figure 16[13].

---

[13] The vertical Order arrow from the top does not portray an information flow but only describes the logical assignment of an order to an activity. A similar argument applies with respect to the vertical Resource arrow from the bottom.

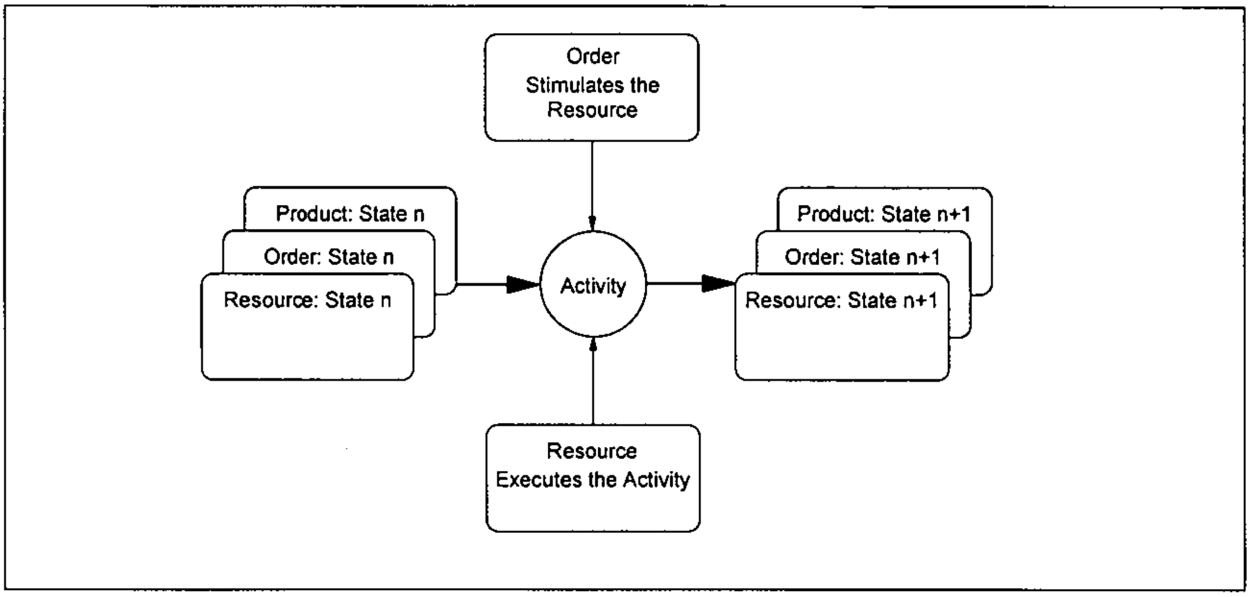


Figure 16: The Generic Activity Model (GAM).

The implementation dependent part of the workflow model of IEM describes a set of interrelated co-operating IEM objects. Each IEM object has specific descriptive features, a life cycle and a set of functions that are inherited. These IEM objects can be further specialised into industry or company specific objects, as illustrated by Figure 17.

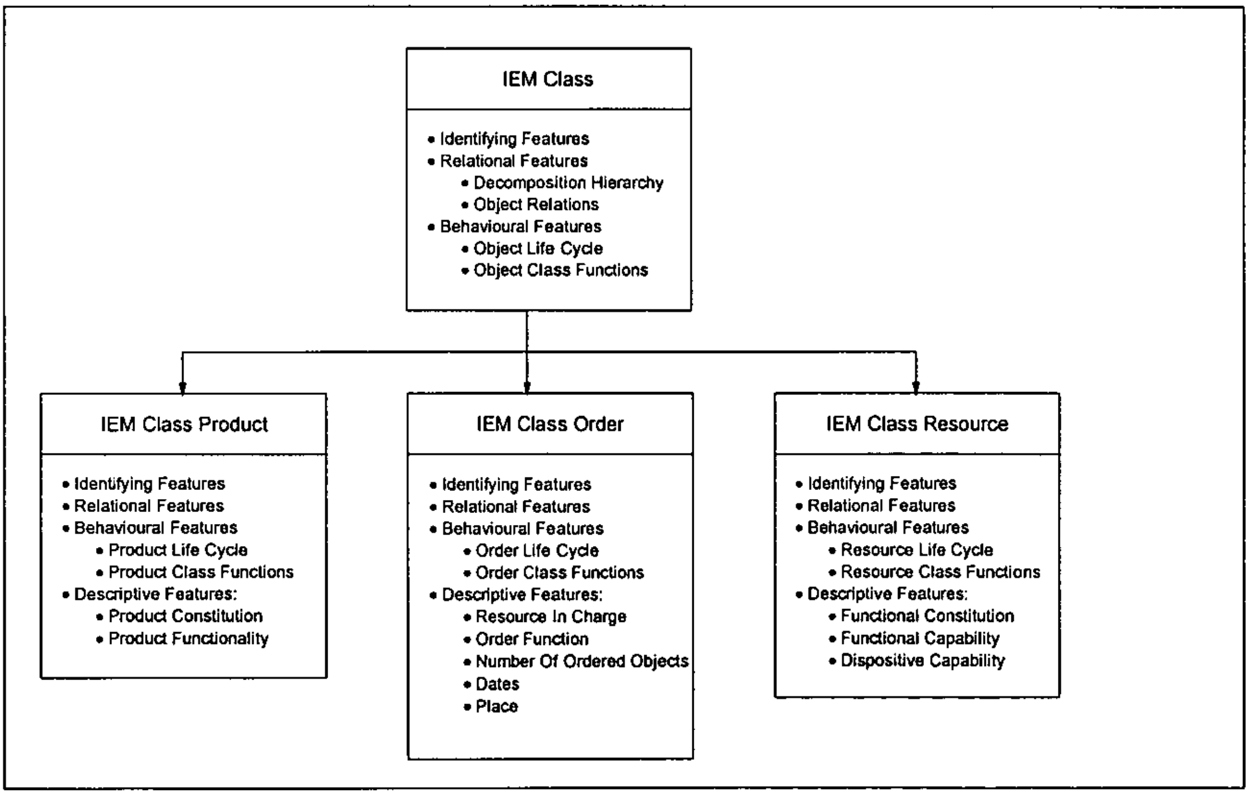


Figure 17: Implementation workflow model.

The function of an IEM object is to manipulate descriptive features and the life cycle (i.e. state and state transitions) of the object. An IEM object can interact with other IEM objects by means of their relational features. When logically an activity is executed technically the objects exchange a series of messages[14].

2.1.3 GRAI Nets

A GRAI Net consists of a description of a set of activities that are ordered by sequential, fan in and fan out links (as illustrated by Figure 18) [Pun84]. Two kinds of activities are recognised by GRAI Nets:

1. Execution activities. An execution activity is a data and/or material transformation activity. Such an activity is defined by the 4-tuple $<\eth, q_0, x, q_f>$ where q_0 is the initial state of an activity, q_f is the final state of an activity, $\eth$ is the transformation function and x represents one or several activity supports.
2. Decision making activities. A decision making activity is an activity that generates a decision. Such an activity is defined by the 6-tuple $<\eth_d, q_0, x, q_f, obj, vd>$ where $\eth_d$ is the decision-making activity process, and obj are the objectives and vd the decision variables involved in reaching a decision.

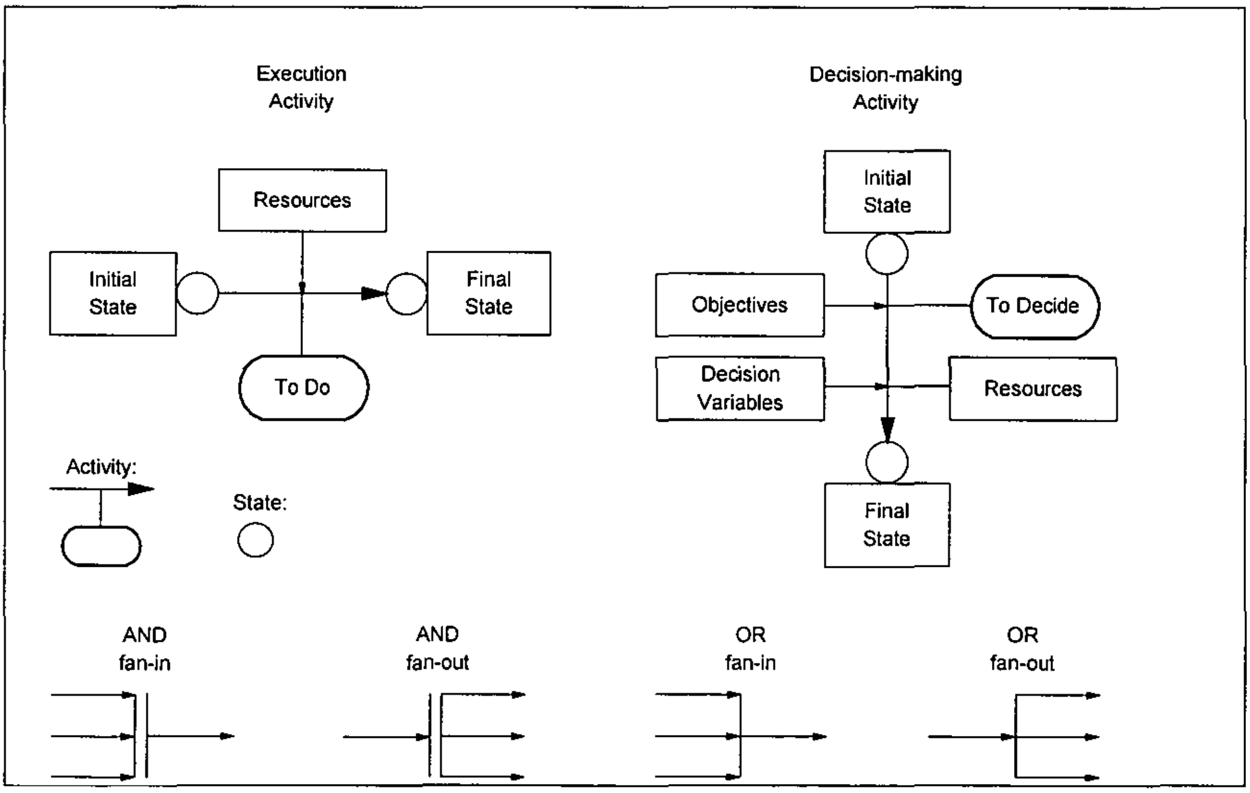


Figure 18: Elementary constructs of a GRAI Net.

The input state of a GRAI Net activity defines key properties of input objects and triggering conditions related to that activity. The transformation function $\eth$ defines how an input state is transformed into an

---

[14] Which messages are exchanged between which objects and how the communication between objects can be described are beyond the scope of current IEM methods.

equivalent output state by either an algorithm or by another GRAI Net (thus enabling functional decomposition).

2.1.4 CIM-OSA

CIM-OSA views an enterprise as a federation of communicating domains [AMI93, Ver96]. A CIM-OSA domain describes a part of an enterprise (but does not necessarily map to a part of the organisational structure). Interaction between two domains is defined by a domain relationship and consists of events and object views that each domain produces and uses (as illustrated by Figure 19). A CIM-OSA event describes a real world happening (such as the generation of an order or the occurrence of a machine failure) that relates to information. A CIM-OSA domain has objectives and constraints that are measurable.

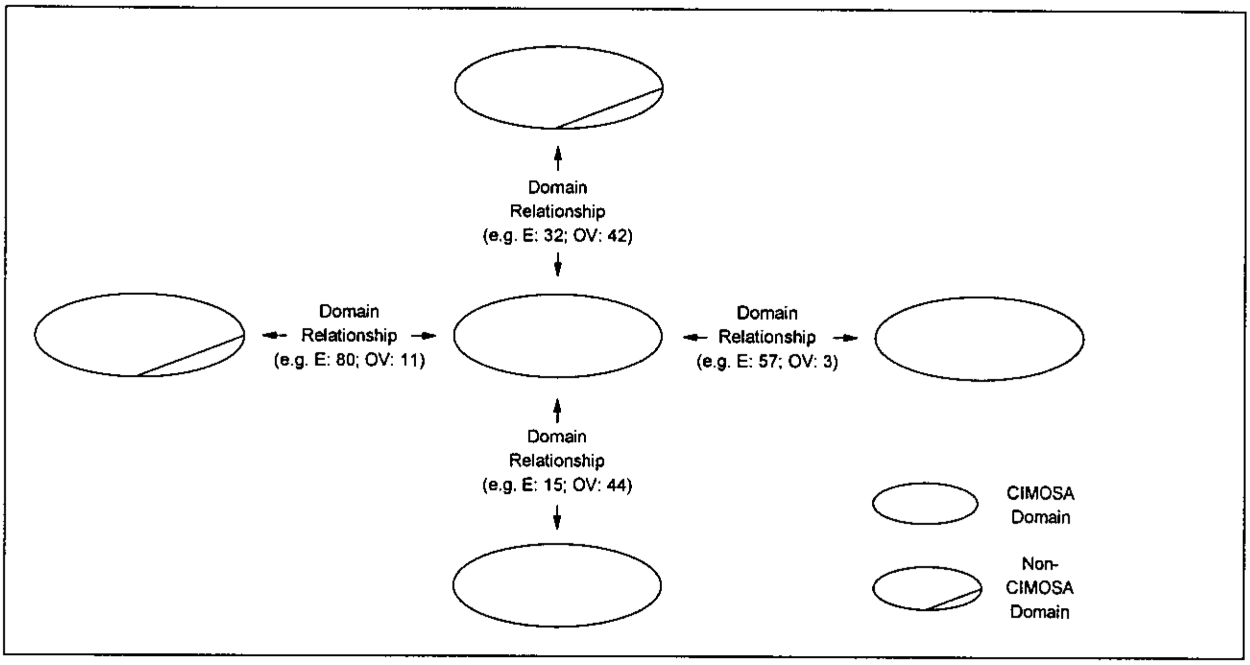


Figure 19: Domains and Domain Relationships.

A CIM-OSA domain consists of a set of domain processes, as illustrated by Figure 20. A domain process consists of a set of functional entities. These functional entities are linked by procedural rules and their execution can only be triggered by an event. Domain processes themselves are not linked to each other by procedural rules and therefore are essentially isolated units of work. For every CIM-OSA event a domain process must exist but an event may be linked to multiple domain processes. A functional entity may be either a business process or an enterprise activity. In CIM-OSA terms a business process is a set of functional entities that are linked by procedural rules. In this way a hierarchy of enterprise activities can be established, as illustrated by Figure 21.

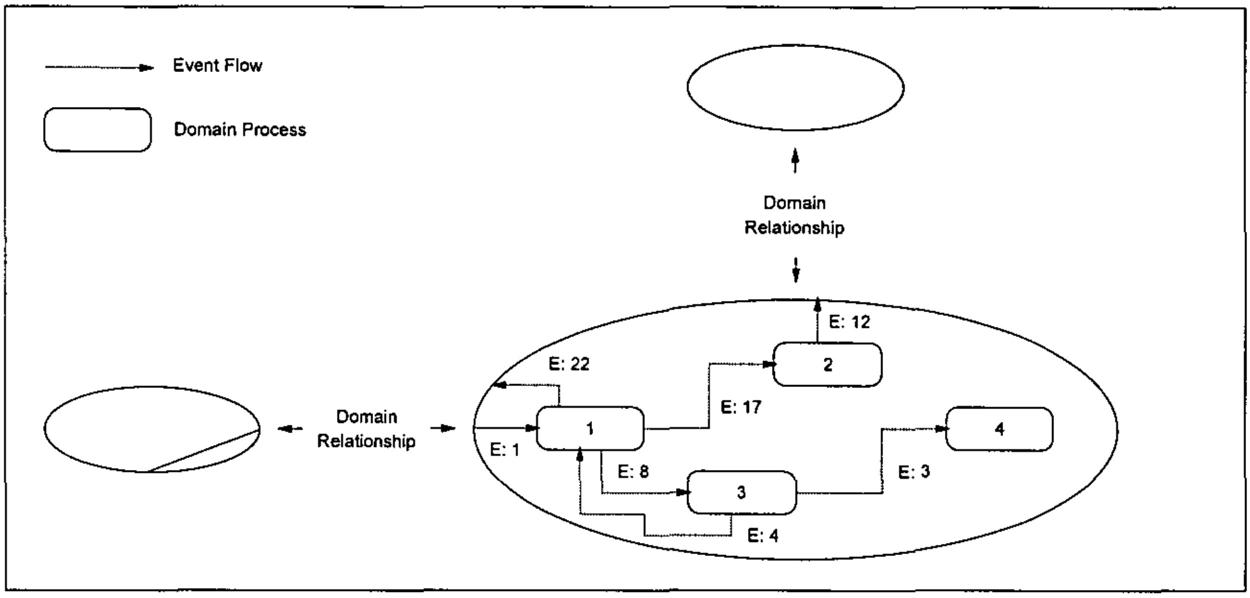


Figure 20: Domain Processes of a Domain.

An enterprise activity is an elementary enterprise task or operation, i.e. CIM-OSA enterprise activities correspond to the lowest level of functional decomposition. An enterprise activity takes as its input the output of a previously performed activity (that is expressed as an object view). A control input is used to control or constrain the execution of an activity. Also an activity may generate a control output. Thus flows of information can be derived from data use/produce relationships connecting CIM-OSA enterprise activities. In CIM-OSA terms an object view consists of information elements (including basic elements: e.g. integer, boolean, and complex elements: e.g. array, list) and other object views. An activity may generate an event that has influence within or outside a domain. Furthermore CIM-OSA enterprise activities have an ending status that consists of a set of termination states.

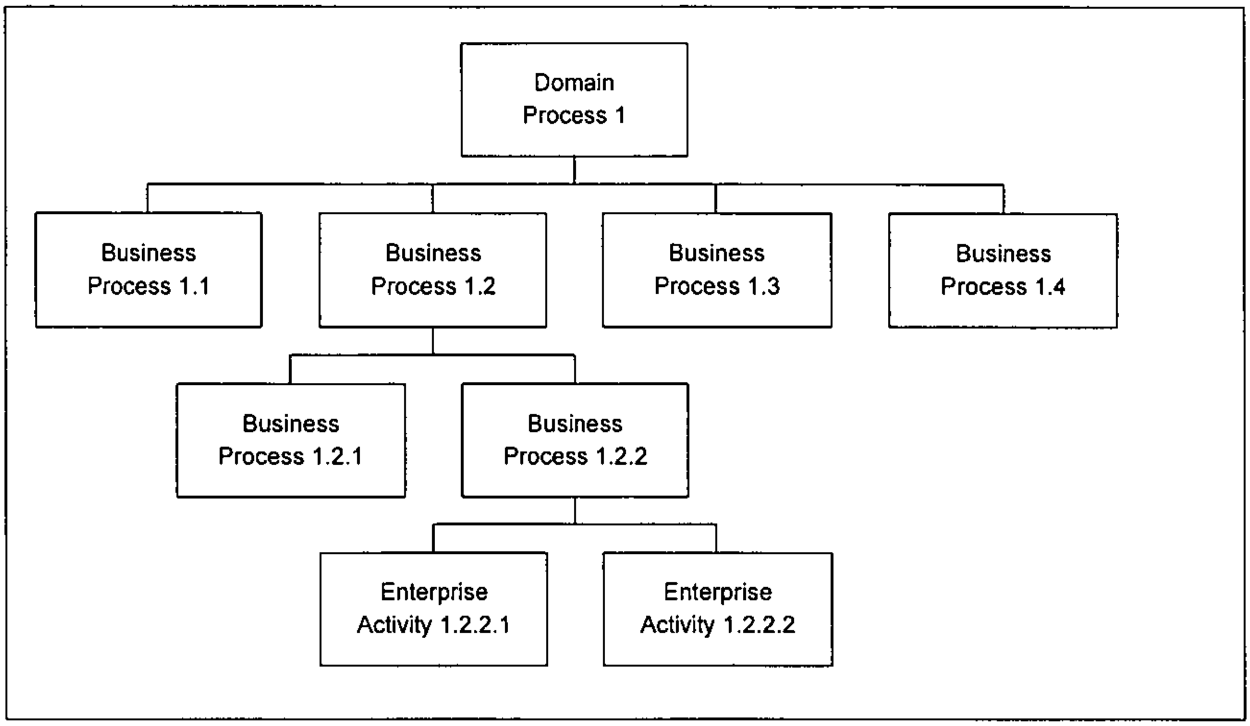


Figure 21: Functional Decomposition of a Domain Process.

The general form of CIM-OSA procedural rules is as follows: WHEN(triggering condition) DO action. Relevant triggering conditions and actions are derived from general objectives and constraints.

Five types of CIM-OSA procedural rules have been defined as follows:

1. Type process triggering: this type of procedural rule is used to trigger the re-execution of a domain process: WHEN (Event-A) DO EF_y, or WHEN (Start) DO EF_y.
2. Type forced: this type of rule is used to activate the next business process or enterprise activity in a manner that is independent of any status information: WHEN(ES(EF_x) = any) DO EF_y.
3. Type conditional: this type of rule is used to activate the next business process or enterprise activity in a manner that depends upon status information: WHEN(ES(EF_x) = value1) DO EF_y.
4. Type spawning: this rule is used to activate two or more defined paths: WHEN(ES(EF_x) = value1) DO EF_1 & EF_2 & EF_3.
5. Type rendezvous: this rule waits for all paths activated to complete and then activates one or more subsequent paths.

CIM-OSA recognises two types of exceptions namely: (1) normal exceptions that are handled by procedural rules referring to a particular ending status, and (2) abnormal exceptions the occurrence of which had not be foreseen and catered for in advance and therefore requires a process to be suspended and decisions to be made elsewhere.

2.1.5 IDEF3

An IDEF3 process schematic consists of units of behaviour, precedence links, junctions, referents and notes (as illustrated in Figure 22) [May92]. Units Of Behaviour (UOB) are either activities or other IDEF3 process schematics. Precedence links express temporal, logical, causal, natural and other precedence relations between units of behaviour. Junctions represent points at which either a single thread in the process diverges into multiple parallel or alternative threads, or multiple threads converge into a single thread. They are the source of multiple precedence links or the destination of multiple precedence links. Further junctions can be either synchronous or asynchronous. Thus junctions are the means to specify the timing and sequencing of UOBs in a process (i.e. the decision logic of a process).

Five types of IDEF3 precedence link exist:

1. Simple precedence link: An instance of source UOB A can occur without an instance of destination UOB B and an instance of UOB B can occur without an instance of UOB A.
2. Constrained precedence link 1: An instance of source UOB A must be followed by an instance of destination UOB B. However instances of UOB B alone are not prohibited.
3. Constrained precedence link 2: An instance of destination UOB B must be preceded by an instance of source UOB A.
4. Constrained precedence link 3: (A) An instance of source UOB A must be followed by an instance of destination UOB B, and (B) an instance of destination UOB B must be preceded by an instance of source UOB A.
5. User-defined or relational links (dashed links). A relationship between a source and a destination UOB which semantics is user-defined (specified in an elaboration document).

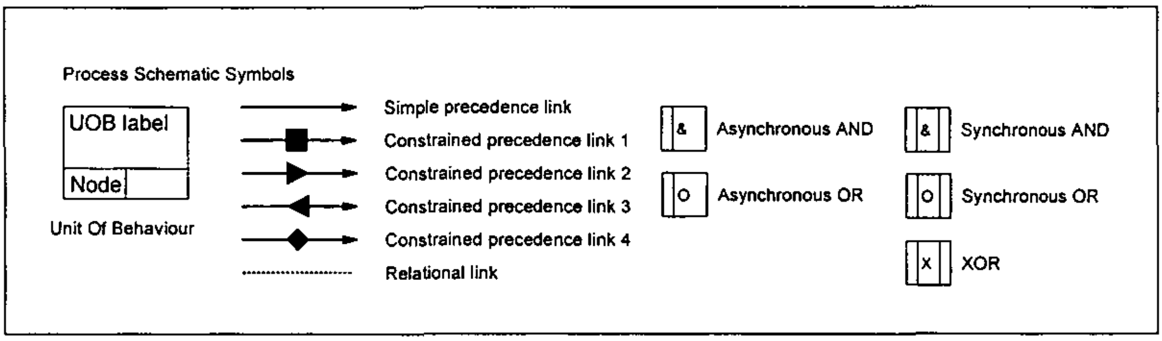

Process Schematic Symbols
UOB label
Node
Unit Of Behaviour
Simple precedence link
Constrained precedence link 1
Constrained precedence link 2
Constrained precedence link 3
Constrained precedence link 4
Relational link
&
Asynchronous AND
O
Asynchronous OR
&
Synchronous AND
O
Synchronous OR
X
XOR

Figure 22: Process schematic symbols.

Consider for example the IDEF3 process specification from Figure 23. Constrained precedence link A defines that UOB 2 'Discuss Committee Reports' must not start before UOB 1 'Call Meeting to Order' has been completed. Simple precedence link B allows UOB 3 'Close Meeting' to be started before UOB 2 has been completed. Constrained precedence link C defines that UOB 4 'Distribute Minutes' must be performed.

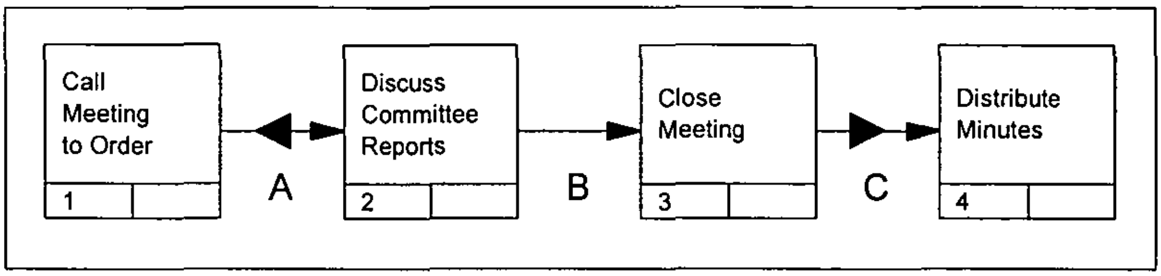


Figure 23: Example IDEF3 meeting process specification.

Four types of junction are specified by IDEF3:

1. Conjunctive fan-out. A point where a single thread is split into multiple parallel threads.
2. Disjunctive fan-out. A point where a single thread is split into alternative or mutually exclusive threads.
3. Conjunctive fan-in. A point where multiple parallel threads are merged into a single thread.
4. Disjunctive fan-in. A point where alternative or mutually exclusive threads are merged into a single thread.

Conjunctive junctions are also termed AND-junctions, and disjunctive junctions are termed OR- (inclusive) and XOR-junctions (exclusive). An OR-junction splits into (or merges) one, many or all threads in a process, whilst a XOR-junction splits into (or 'merges') one thread in a process. Disjunctive junctions (OR- and XOR-junctions) also have an associated decision-logic to determine which of the outgoing precedence links and UOBs will be activated (i.e. how the process will branch), although this is not formally specified in the semantics of IDEF3.

Synchronous junctions dictate that the UOBs of their outgoing precedence links must be started simultaneously, or that the UOBs of their incoming precedence links must be completed simultaneously (for a path to continue). Asynchronous junctions indicate that the UOBs of their outgoing precedence links may start at any time, or that the UOBs of their incoming precedence links do not have to complete at the same time (for a path to continue).

Consider for example the IDEF3 process specification from Figure 24. Synchronous AND junction J1 indicates that instances of UOBs B, C and D will initiate simultaneously after instance UOB A has been completed. Asynchronous OR junction J2 defines that only one of the three threads (composed by B and E, C, and D) needs to complete at some point in time before an instance of UOB F may be started. However for a process to complete all threads need to complete.

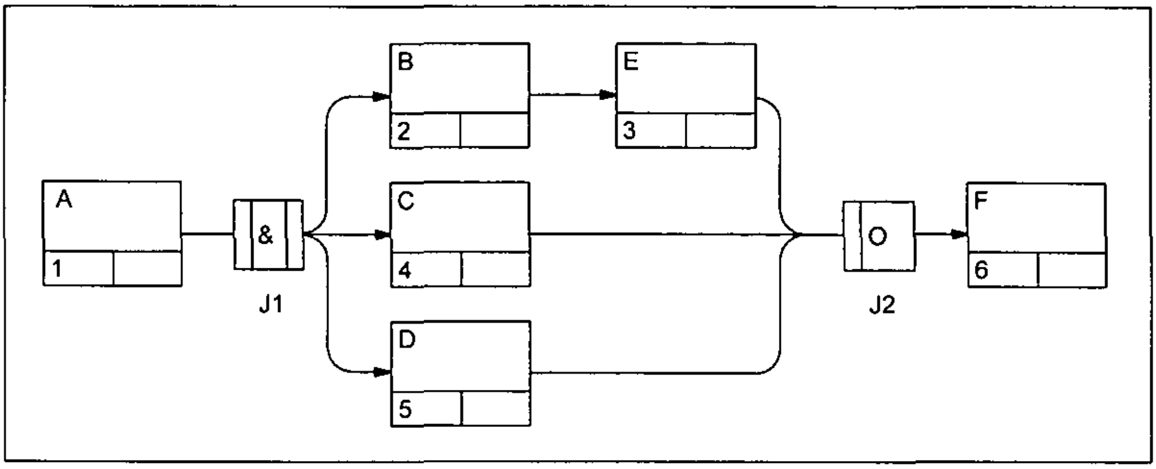


Figure 24: Example process schematic.

2.2 Summary

Workflow models can be created using either SADT/IDEF0, Grai Nets, IEM, CIM-OSA and IDEF3 modelling notations. However the resultant properties of the workflow models so produced will be dependent on relative capabilities of each modelling notation used. Hence it is informative to compare conventional modelling approaches from a number of viewpoints as explained below.

1. Flow of data viewpoint. Using available modelling constructs and notations this viewpoint can typically include reference to the following:
    - 'Data Structure': this concerns the types of data item that an activity can use and produce and the way that these data item types are related to each other (such as within some form of hierarchical or networked organisational arrangement). Typical data items may be strings or integers.
    - 'Data Produced' and 'Data Consumed': this concerns the flow of data from producer activities to consumer activities. This kind of data flow can be represented using either SADT/IDEF0, Grai Nets, IEM, and CIM-OSA notations.
    - 'Data Splitting' and 'Data Merging' rules: these rules determine how unique objects belonging to one flow of data are split into multiple objects stored in different flows of data or to determine how multiple objects from various flows of data can be merged into a unique object represented and stored in one flow of data. Data merging and splitting can be modelled by using the SADT/IDEF0 notation.
2. Flow of control viewpoint. This viewpoint typically references the following kinds of relationship and modelling construct.
    - 'Precedence Relationships': These relationships determine the basic order in which downstream and upstream activities should be executed. Precedence relationships can be formally represented

using IDEF3, IEM and CIM-OSA modelling constructs. Whereas all of these modelling methods recognise the precedence order A then B (i.e. that B may only be executed after A), IDEF3 additionally defines a number of alternative precedence orders.

- 'Merging' and 'Splitting' modelling constructs: Typically these constructs are used to split a single precedence link into a number of precedence links (fan-out) or to merge a number of precedence links into a single precedence link (fan-in), whilst satisfying a particular set of conditions. Examples of merging and splitting modelling constructs are provided as an integral part of the IDEF3, IEM and CIM-OSA notations. Furthermore AND, OR and XOR logic is assigned to these modelling constructs and infers the following meaning. An activity may only be started if all incoming precedence links are enabled (AND), if some of them are enabled (OR) or precisely one is enabled (XOR).
- Specification of the 'Timing of Merging and Splitting' operations: synchronisation conditions related to the start or completion of multiple activities can be specified so that workflow activities can be executed synchronously or asynchronously. IDEF3 notation provides a set of modelling constructs that can be used for this purpose.

3. 'Post Condition' relationships: Post conditions are used to determine whether or not an activity execution is successful. Grai Nets and IDEF3 provide modelling constructs to encode post condition relationships.

Table 2 indicates whether or not appropriate coverage of the viewpoints described above is provided by the current generation of enterprise modelling notations.

	Flow of Data			**Flow of Control**			**Post Conditions**
	Data Structure	**Data Produce and Use**	**Data Splitting and Merging**	**Precedence Relationships**	**Merging/Splitting**	**Timing Merging/Splitting**	
SADT/ IDEF0	Names for input and output objects. But no data types or means of encoding relationships between data items.	Yes. Flow of data into and out of ICOM boxes can be represented.	Yes. Branch, join and arrow constructs provided.	No.	No.	No.	No.
Grai Nets	Names for input and output objects. But no data types or means of encoding relationships between data items.	Yes. Fan in and fan out constructs are provided.	No.	No.	No.	No.	Partially. Objectives and decision variables are provided.
IEM	Names. But no specific predefined data types are defined. Nor are relationships between data items defined.	No.	No.	Yes. Supports sequential concatenation.	Partially. Parallel concatenation and other alternative concatenation approaches are supported.	Not explicitly defined. Thus either synchronous or asynchronous timing is inferred.	No.
IDEF3	No.	No.	No.	Yes. Various types	Yes. Fan in and fan	Synchronous and	No.

				of precedence relationships are supported.	out junction conditions can be encoded via merge/split types, i.e. AND/OR/XOR.	asynchronous modelling constructs are provided.	
CIM-OSA	Names for input and output objects. But no data types or means of encoding relationships between data items.	Yes. Flow of data amongst activities is provided.	No.	Yes, it is possible to define parallel and sequential paths of activity execution.	Yes. Fan in and fan out junction conditions can be encoded via merge/split types, i.e. AND and OR.	Synchronous and asynchronous modelling constructs are provided.	No.

Table 2: Coverage of viewpoints by the current generation of enterprise modelling notations.

3.0 Event-Condition-Action (ECA) workflow approaches

Event-Condition-Action (ECA) workflow approaches can be used to order the execution of activities by means of event-condition-action rules [Luc95a, Luc95b, Wid96, Bar96]. They enable the behaviour of a system to be defined in terms of (1) events that represent occurrences of relevance in a system, (2) conditions that define possible states of the system and (3) actions that realise state-changes in the system. Event-Condition-Action modelling notations were considered to be a necessary prime force of study in this research project since their potential use can trigger different actions when particular workflow system states are achieved and when certain well defined events occur. ECA rules had previously been extensively researched within the context of active database systems which are considered to be an enhanced form of traditional, passive database systems[15]. Whereas traditional database systems only respond to actions triggered by users or application programs, active database systems can monitor changes in the state of a database and automatically trigger suitable actions themselves. According to [Wid96] potentially the rule processing capabilities of ECA approaches may also serve workflow applications well. To date ECA modelling research has focussed mainly on the following three issues [Wid96]:

1. Rule expressiveness or rule language development: this is necessary to adequately represent events, conditions and actions that need to be specified.
2. Rule execution semantics: namely the correctness of interleaving of parts of event-condition-action rules.
3. Rule execution efficiency: concerning the efficient evaluation of large sets of rules and an optimisation of execution conditions.

Active database systems research and the development of ECA modelling techniques were considered important areas of database research during the late 1980's and early 1990s.

The following subsections consider key features of well documented ECA approaches.

3.1 Review Event-Condition-Action workflow approaches

3.1.1 WIDE

The WIDE active database system is an object-oriented database system that can be operated on by Event-Condition-Action (ECA) rule systems [Bar99a]. Objects store information about the state and outcomes of tasks (i.e. activities) and workflows (i.e. ordered sets of activities). ECA rules may refer in their event,

---

[15] Expert systems and deductive databases are two other closely related areas of research requiring rule-based processing [Wid96]. However whereas active database systems are event-driven, expert systems and deductive databases are condition-driven.

condition, or action part to objects stored in the database. Within the context of handling workflow exceptions (A) events denote the possible occurrence of an exceptional situation, (B) conditions identify the state of work activities and (C) actions perform the operations needed to handle an exception for a given state of a workflow.

A number of event types are supported by WIDE, namely:

1. Data change events. Events that are raised when changes occur to data stored in a database. Data change events include createObject(Object), modifyObject(Object.Attribute), and delete-Object(Object).
2. Workflow events. Events that are generated when a case or a task is started or completed. Workflow events encompass caseStart(activityA), caseEnd(activityA), taskStart(workflow1), and taskEnd-(workflow1).
3. Temporal events. Events that are raised upon the occurrence of defined temporal instants. Temporal events for example include caseTemporalDelay(ActivityA) and taskTemporal-Delay(workflow1).
4. External events. Events that are raised by users or other applications. External events are defined by raise("ExternalEvent").

A condition consists of a set of predicates that refer to states of a database. When a condition is evaluated the state of its predicates refer to the actual state of the database. If a condition is satisfied the action part is executed, otherwise it is not. Alternative predicate values can draw comparison between expressions where an expression can be an object variable or a constant. An example predicate may be T.executor = "Lisa". Furthermore a condition contains additional statements that carry objects selected by the evaluation of conditions linked to the action part (so called bindings). For example consider the binding: case(C), occurred(caseStart, C) for a caseStart event. This declaration means that upon some caseStart event variable C will refer to this case. Subsequently the variable C can be used in the action part.

Actions can be 'data manipulation actions' or 'workflow actions'. Examples of data manipulation actions are object create, delete and modify actions. Examples of workflow actions are to start, rollback, or suspend the execution of cases and tasks, or to delegate the execution of tasks, etc.

Consider for example the event-condition-action rule illustrated by Figure 25.

```
define trigger   missingPayment
events           caseEnd
condition        case(C), occurred(caseEnd, C), roomReservation(R), R.caseId = C,
                 R.paymentDone = FALSE
Actions          startCase("cancelBooking", R.customerId)
```

Figure 25: Example event-condition-action rule.

This rule will be processed by WIDE as follows. Firstly when a caseEnd event occurs, the case is assigned to variable C of type case (by means of predicates case(C) and occurred(caseEnd, C)). Next the room reservation object is retrieved for this case and assigned to variable R of type roomReservation (by means of predicates roomReservation(R) and R.caseId = C). Finally a check is performed as to whether the customer has made the required payment (by means of predicate R.paymentDone = FALSE). If the condition evaluates to true then the action startCase("cancelBooking", R.customerId) is executed. As in the condition part R is bound to some object this object can be referred to in the action part.

3.1.2 Rapide

The Rapide system architecture comprises components and connections as illustrated by Figure 26. A component has an interface and a module. The interface of a component defines the actions[16] other components may invoke (i.e. public actions) in respect to this module and the actions a module may invoke on other components (i.e. extern actions). Rapide interface definition is illustrated by Figure 27. A module implements the public actions an interface defines and its code can be written in an arbitrary programming language. Connections link extern actions (provided by one interface) to public actions (provided by another interface) and thus define how events generated at some interface cause events to be received at other interfaces. When an extern action is invoked the parameters of this action are mapped onto the parameters of the public action. In this way means of achieving data flow and synchronisation between modules is provided.

The actions executed at a component result in a partially ordered set of events (also termed a poset) representing the dependencies between events. An event B depends on an event A (A → B) if A has happened before B.

---

[16] Rapide actions represents asynchronous communication between components. Whereas in Rapide a function models synchronous communication.

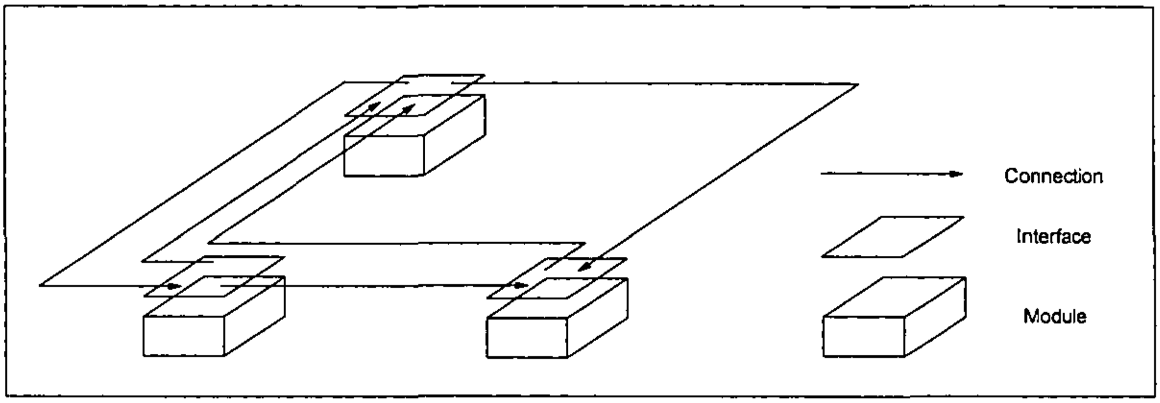


Figure 26: The system architecture of Rapide.

Associated with a Rapide interface is a behaviour and some constraints. The behaviour of an interface is defined by triggers and their actions. A trigger is a sub set of events from the poset generated at a component that meets particular criteria, i.e. an event pattern. Triggers are generated by the poset at components and triggers act to carry out a set of operations. When the action is executed a new poset results. A constraint is an event pattern used to check whether sets of partially ordered events adhere to a particular behaviour. If such a constraint is violated this is signalled to the user.

With Rapide it is possible to analyse the behaviour of the system before the system is built, i.e. before modules are written. In this way the system architecture can be validated. If constraint violations occur during testing either certain mappings are incorrect or an instance of a mapping does not conform to the reference architecture.

An interface defines the actions

type Application **is interface**

  extern action Request (p: params);

  public action Results (q: params);

end Application

Figure 27: An interface definition.

Pattern :: =

  basic_pattern

  | pattern binary_pattern_operator pattern

  | pattern '^' '(' iterator_operator binary_pattern_operator ')'

  | pattern **where** guard

  | { placeholder_check } pattern

| '(' pattern ')' | **empty** | **any**

| pattern **during** '(' expression ')' ?

Binary_pattern_operator ::= '->' | '||' | **'and'** | **'or'** | '~' | '<->'

Iterator_operator ::= '*' | '+' | expression

Guard ::= boolean_expression

A basic_pattern is simply the name of an action.

Pattern operators have the following informal semantics:

- Dependent: P -> P': all of the events that match P' depend on all of the events that match P.
- Independent: P || P: a match of patterns P and P' where none of the events that match P are dependent on any of the events that match P'.
- Both: P **and** P': a match of patterns P and P'.
- Either: P **or** P': a match of pattern P or a match of pattern P'.
- Distinct: P ~ P': a match of patterns P and P' where all of the events that match P are distinct from the events that match P'.
- Equivalent conjunction: P <=> P': a match for P that is also a match for P'.
- Iteration: P ∧ (iterexp op) where iterexp is one of * (zero or more), + (one or more), or n (exactly n) copies of P, each copy being related to the others by op which is any of the preceding binary operations.
- Restriction: P **where** E: a match of pattern P where the boolean expression E is true; the expression is evaluated when the last of the events matching P is received.
- Temporal restriction P **during** (m, n): a match of pattern P where all of the events of the match start and finish within the time interval m to n.

3.2 Summary

With reference to their potential use in specifying and enacting robust workflows Event-Condition-Action workflow approaches such as WIDE and Rapide possess key properties as follows (as illustrated by Table 3):

1. Model Types: Three kinds of event-condition-action types can be supported [Wid96], namely: (A) E-C-A where the event, condition and action parts are separate and readily configurable, (B) EC-A where the event part is implicitly linked to the condition part but separated from the action part, and (C) E-CA where the condition part is an integral part of the action part but where this part is separated from the event specification. As Rapide only recognises one kind of event (i.e. the completion of an action) it can be classified as EC-A type. Whichever action completes, the conditions of all rules will be evaluated. Yet Rapide is capable of triggering different actions for various conditions. WIDE is based on an E-C-A type since (A) it recognises many different types of event and (B) it can link customised actions to various conditions.
2. Event Types: Namely the kinds of occurrence that trigger the execution of an ECA rule. All event-condition-action workflow approaches support the completion of a unit of work as an event occurrence. In the case of Rapide this occurs following the completion of an action and for WIDE this occurs when a case completes. Rapide does not support many other types of event but WIDE does as it supports data modification events, temporal events, workflow events, etc.
3. Condition Types: Condition types can be characterised by the kinds of construct used to determine conditions under which an activity is performed. Apparently two main types of techniques are used to determine particular conditions and thereby determine the status of a workflow. The first type of technique checks the status of a sub set of activities that previously have been performed, i.e. it adopts a process-centric workflow-state view. The Rapide approach is based on the use of such a view where it provides a capability to check for patterns of previously performed actions. The second type of technique references the values of object variables manipulated by activities, i.e. it adopts a data-centric workflow-state view. The WIDE approach uses this second type of technique.
4. Action Types: Action types are distinguished by the type and the number of steps taken if some condition is met. Rapide only supports the execution of one unit of work (i.e. an action) as a next step. However WIDE can initiate multiple units of work (i.e. cases) and can rollback and suspend the execution of multiple units of functionality.

Approach	**Model Type**	**Event Types Supported**	**Condition Types Supported**	**Action Types Supported**
Rapide	EC-A	Completion of an action. Events are not explicitly modelled but are encoded as part of a condition.	Patterns of previously performed actions. Actions can be dependent, independent, both, either, distinct, equivalent conjunction, iteration, restriction and temporal restriction.	Start an action.
WIDE	E-C-A	Data modification events, temporal events, workflow events and external events	Predicates defined with respect to object variables	Start, suspend and rollback cases.

Table 3: Summary of Event-Condition-Action viewpoints.

4.0 Transactional workflow approaches

Advanced transaction approaches have been developed by the database and transaction community to support the needs of non-traditional database applications such as office automation, CAD/CAM and software engineering. Advanced transaction approaches can also be deployed to specify and enact certain aspects of workflow systems. The responsible research community recognised the need to adapt and advance conventional transaction processing concepts and techniques in order to co-ordinate the execution of domain activities [Alo97a, Du97, Kam98, She96]. A principal aim has been to preserve the consistency of databases shared by concurrently executing activities [Alo96, Bre93, Kam98]. A number of classes have been developed, namely: (A) extended transaction approaches, (B) relaxed transaction approaches and (C) co-operative transaction approaches [Kam95]. Extended transaction approaches are essentially based on early traditional transaction processing philosophies and come with an underpinning theoretical framework. However they have been found to be 'too restrictive' in certain domains [Kam98]. A second generation of advanced transaction approaches, i.e. relaxed and co-operative transaction approaches, focus less on the development of such a theoretical framework but concentrate on the selection and elaboration of elementary techniques found to suit a particular application purpose. Indeed a practical need to develop specific approaches for different application domains had been increasingly recognised [Moh95, She96]. Although this previous literature has recognised some aspects of workflow system requirements, little has been explicitly written about them. Indeed the requirements of workflow applications have thus far been stated in terms of already developed techniques rather than being described from a logical point of view. Further, as yet, few advanced transaction approaches have been implemented in the form of commercial transaction management software or have been used to underpin the operation of commercial application software [Alo97a, Moh95]. It may be that advanced transactional techniques are viewed as being suitable for database applications only and cannot readily be successfully applied in manufacturing domains.

4.1 Review transactional workflow approaches

4.1.1 Sagas

A Saga is a long lived transaction[17] composed of an ordered set of elementary transactions and/or other Sagas [Gar87a, Gar87b, Gar91]. Whereas operations that are performed by elementary transactions operating on data items may not be interleaved with operations carried out by other elementary transactions, Saga transactions themselves may be interleaved. Thus in Saga terms elementary transactions are indivisible units of work. As a Saga may be embedded into another Saga they support recursion or the nesting of transactions. For example transactions at level t-2 make up a Saga at level t-1. A Saga together with other Sagas and transactions at level t-1 can be combined into a single Saga at level t. Further

---

[17] In business terms, an activity can be viewed as being the equivalent to a transaction.

elementary transactions last only a fraction of a second (e.g. hundreds or tens of a second) while a Saga may last many hours, days or weeks (but are completed within a finite time).

In the approach adopted by Sagas an exception is considered to be the result of a negative outcome when performing an activity. Under conditions where an activity cannot successfully complete an exception occurs. If such a situation arises then Saga systems compensate for all previously performed activities. To amend the results of a previously performed activity A, a compensating transaction C is carried out. The compensating transaction C undoes, from a semantic point of view, any of the actions performed by A but does not necessarily return the database to the state that existed prior to the execution of A. Thus either a set of activities A_1, A_2, $A_{...}$, A_n is completely executed, or those activities that have been performed prior to the exception are compensated for by carrying out a set of undo activities (namely C_n, $C_{...}$, C_2, C_1) which are carried out in a reverse order compared to that of the original activities. If an activity is itself a Saga first all activities within the Saga must be compensated for. The system can automatically generate compensation activities for the set of previously completed activities. However the rather draconian approach of Saga systems is clearly not required in most real operating environments. On many occasions only a sub set of activities will become invalid. But a Saga approach to compensation is justified when a workflow is cancelled, under which conditions the actions of previously performed activities must be reversed.

Although it has been argued that the Saga approach is not really viable in real operating environments, Saga provided an early way of formally defining a correct execution sequence for activities in environments where exceptions may occur. To understand its objective and conception the approach must be viewed from the starting point context of traditional transaction approaches. Traditionally transactions were considered to hold on to data items for the period of their processing time (i.e. so called ACID-transactions). As workflows are also units of processing a problem resulted that workflows hold on to data items for long periods of time causing other workflows (that require access to the same data items) to abort, or to wait for unacceptable long periods of time. Thus the Saga approach of releasing data items upon the completion of an activity was a useful first step on a migration path from the use of 'data oriented transaction approaches' to 'process-oriented transaction approaches' [Bre93]. The Saga requirement to compensate for all previously performed activities upon the occurrence of an exception stems from the requirement of atomicity.

4.1.2 ConTracts

A ConTract is a consistent execution of an arbitrary sequence of predefined actions (called steps) according to an explicitly specified control flow description (called a script) [Wae92, Sch93, Reu95]. A step is an elementary unit of work and constitutes a basic computation for an application. A script describes the control flow of a long-lived activity and is modelled by typical modelling concepts and constructs such as sequence, parallel, branch and loop modelling elements. A ConTract control flow specification can be linked to sets of activities that must be executed as an atomic unit. A first advantage of distinguishing between scripts and steps, is that steps can be re-used in other scripts. A second advantage is that the task of analysing and implementing systems has been much simplified as a number of issues can now be separated, namely: (1) the definition of logical failures associated with steps, (2) the definition of atomic units of work, in the case of a logical failure of a step, and (3) the definition of a standard flow of control, in the case where an activity completes successfully.

The approach as defined by ConTracts was developed to facilitate compensation of a selected set of activities under conditions where an exception has occurred and where an activity cannot be completed [Sch93]. Steps from a script are grouped into sets which must be executed as an atomic unit of work. If one activity from such a set cannot complete (i.e. an exception or a logical failure occurs) all previously performed activities from this set must be compensated for and another step may be started in order to resolve the exception. An example ConTract scenario is illustrated by Figure 28. In this way application logic is modelled. An atomic unit of execution can be defined by the following type of declaration: T (S1, S2, S..) and DEPENDENCY(T abort -> begin SX).

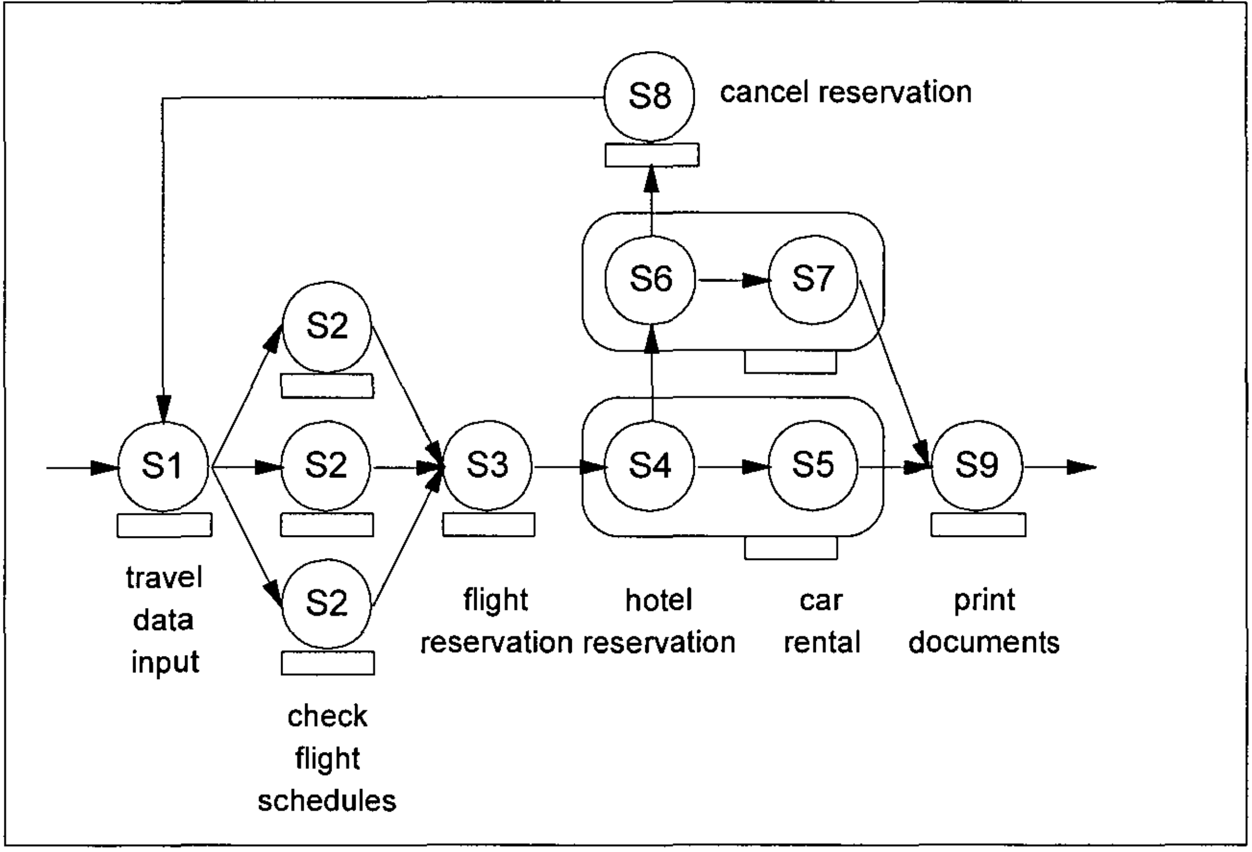


Figure 28: Example ConTract Scenario.

Consider for example a business trip workflow which consists of typical activities such as flight consulting S2, flight booking S3 and hotel and car booking S4 and S5. Activities S4 and S5 form an atomic unit of work, as will activities S6 and S7, as indicated by the dotted lines around (S4, S5) and (S6, S7). The three activities S2 are performed in parallel, in order to select the best flight for the customer. The flight is then booked by activity S3. Next an attempt is made to book a hotel and a car, e.g. at hotel ABC and from rental company XYZ (as these give an exclusive discount to each other and are therefore only booked in combinations). If either of these last two activities fail to complete, all previously performed activities within the atomic unit of work need to be compensated for. For example if a hotel room has been booked but no car is available the booking for the room is cancelled. This example corresponds to a logical failure of an activity and to resolve this situation it is necessary to select another hotel/car rental company combination. A generalisation of the business trip example shows that the ConTracts approach provides a widely applicable means of handling logical failures of activities.

4.1.3 Partial Rollbacks

In Partial Rollbacks terminology, a process is a set of activities that are ordered by means of 'control connectors' [Ley95]. An activity can be either a 'sub process', a 'block' or a 'program'. A sub process is itself a process model. A block is a sub process which is repetitively executed until its exit condition is fulfilled. Programs represent the basic execution steps within a process model. An exit condition (i.e. a predicate) determines whether the execution of an activity has been successful. Activities are ordered by means of sequence, loop, branch, fork and join control connectors. A control connector can have a predicate (i.e. a transition condition) that determines whether a connector is enabled. Control connectors are part of a boolean expression that determines whether an activity may be executed (i.e. a so called activation condition).

The flow of data between activities is defined by means of activity input/output containers and data connectors (as illustrated by Figure 29). An activity input container defines the input parameters of an activity and an activity output container defines the output parameters of an activity. A data connector links a consumer activity to a producer activity and defines via use of a container map which output parameters of a producer activity map to which input parameters of a consumer activity. In this way different names for input/output parameters can be used.

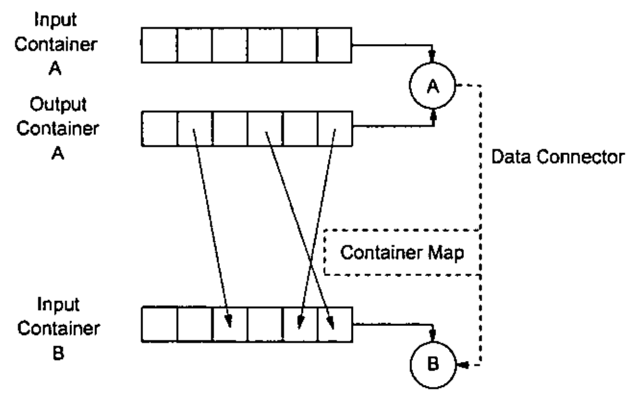


Figure 29: Illustration of Partial Rollbacks containers and connectors.

A sphere consists of a sub set of activities from a process specification that (1) need to complete successfully (if activated) or (2) are required to be compensated and/or restarted in case the execution of one of them should fail. Multiple spheres may be specified on top of a process specification and spheres may overlap. Further a sphere contains another sphere if (1) a sphere contains sub process activities and/or block activities and (2) the processes these activities consist of contain a sphere. A sphere does not necessarily have to induce a connected sub graph within the process specification. Consider for example the process specification illustrated by Figures 31.A and B. The specifications consist of ordered activities A, B, C, D, E, F, H, J, and K. On top of this control flow specification three spheres have been defined (i.e. sphere 1, 2, and 3) where sphere 1 and sphere 2 overlap with respect to activity D. Spheres 1, 2 and 3 result in a connected sub graph. On the other hand sphere 4 from Figure 31.B does not induce a connected sub graph but it consists of activity set A, C and D and activity set J and K. An example of a sphere that contains another sphere is illustrated in Figure 31.C. Sphere 5 encompasses activity A which is a sub process activity. The sub process of activity A contains sphere 6. In Partial Rollbacks terms the use of spheres transforms a process into a business transaction. Key properties of Partial Rollbacks' spheres and activities and their relationships are outlined in Table 4 and Figure 30.

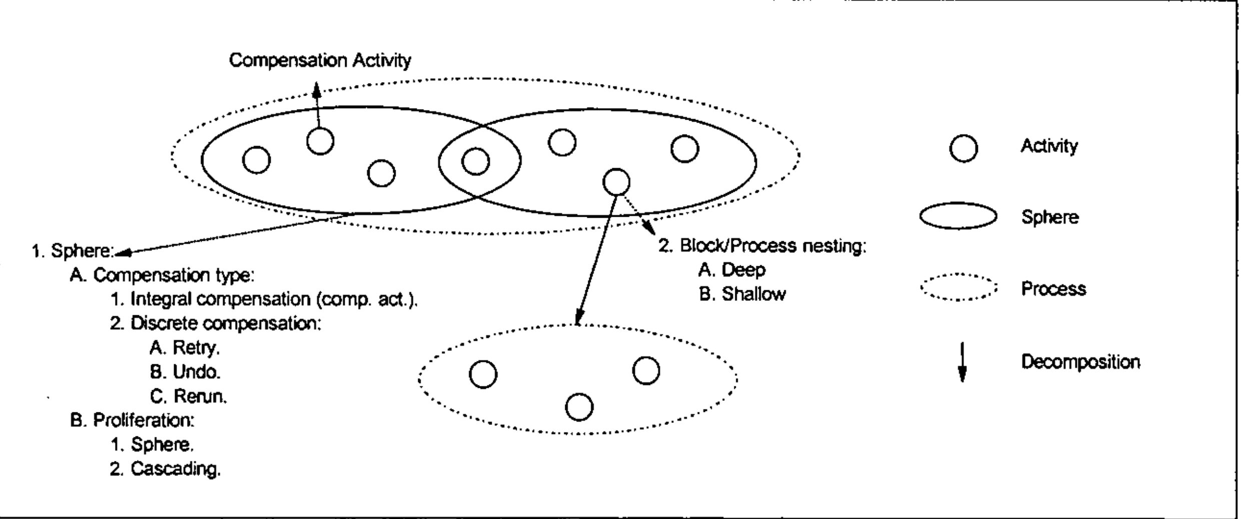


Figure 30: Illustration of Partial Rollbacks' spheres and activities.

1. Sphere:
   A. Compensation type:
      1. Integral compensation. Executes the single compensation activity associated with the sphere.
      2. Discrete compensation:
         A. Retry: Executes compensation activities for those activities actually executed within the sphere, and reschedules all initiating nodes for execution.
         B. Undo: Executes compensation activities for those activities actually executed within the sphere.
         C. Rerun: Reschedules all initiating nodes for execution.

   B. Proliferation:
      1. Sphere: Compensation of the sphere does not trigger compensation of any dependent sphere.
      2. Cascading: Compensation of the sphere triggers compensation of all dependent spheres.
2. Block/Process nesting:
   A. Deep: Executes the compensation activities for activities executed within the block or process.
   B. Shallow: Executes the compensation activity associated with the activity itself.

Table 4: Key properties of Partial Rollbacks' spheres and process nesting.

The Partial Rollbacks approach to activity compensation can be interpreted as follows[18]. Suppose failure occurs for an activity that belongs to a sphere. Sphere compensation is determined as follows. In cases where integral compensation is required a single compensation activity associated with the sphere is executed and no further compensation is performed. In cases where discrete compensation is needed either a 'retry', 'undo' or 'rerun' approach to compensation is adopted. If a retry approach is selected (1) compensation activities are executed for those activities that have been executed within a sphere in the reverse order to their original execution, and (2) all actual initiating nodes[19] are rescheduled for execution. Alternatively if an 'undo approach' is selected only point 1 applies and if a 'rerun approach' is selected then only point 2 applies.

Whatever type of discrete compensation is selected, if an activity is to be compensated and the type of activity is (1) a program then the compensation activity associated with this activity is executed or (2) a process or a block its nesting properties determine the nature of compensation. If its nesting property is defined as being a 'deep compensation' the compensation activities are run for those activities already executed within a process. However if it concerns a 'shallow compensation' the compensation activity associated with the activity itself is run.

Finally handling an exception within a sphere ends by checking its proliferation property. If the property is set to the granularity of a sphere no further action is undertaken even though spheres may be dependent. If it is set to 'cascading', overlapping spheres will need to be compensated as well. If spheres (1) share a common executed activity or (2) data generated by a compensated activity in one sphere has been used by an activity in another sphere, then compensation of the sphere is triggered. Point (2) requires an evaluation

---

[18] A number of features related to compensation (such as for example iterate_compensation, activity_proliferation, backout_protected, and join_backout) have not been discussed as they are not central to the research conducted. Rather the research has focussed on the use of spheres to handle exceptions.

[19] An actual initiating node is an activity triggered by the user or as a result of executing an activity contained in another process.

of data dependencies and actual data flows. The input of a compensation activity is defined by a special container (i.e. a compensation container). A compensation container may be connected to other containers (including the activity's own input and output container).

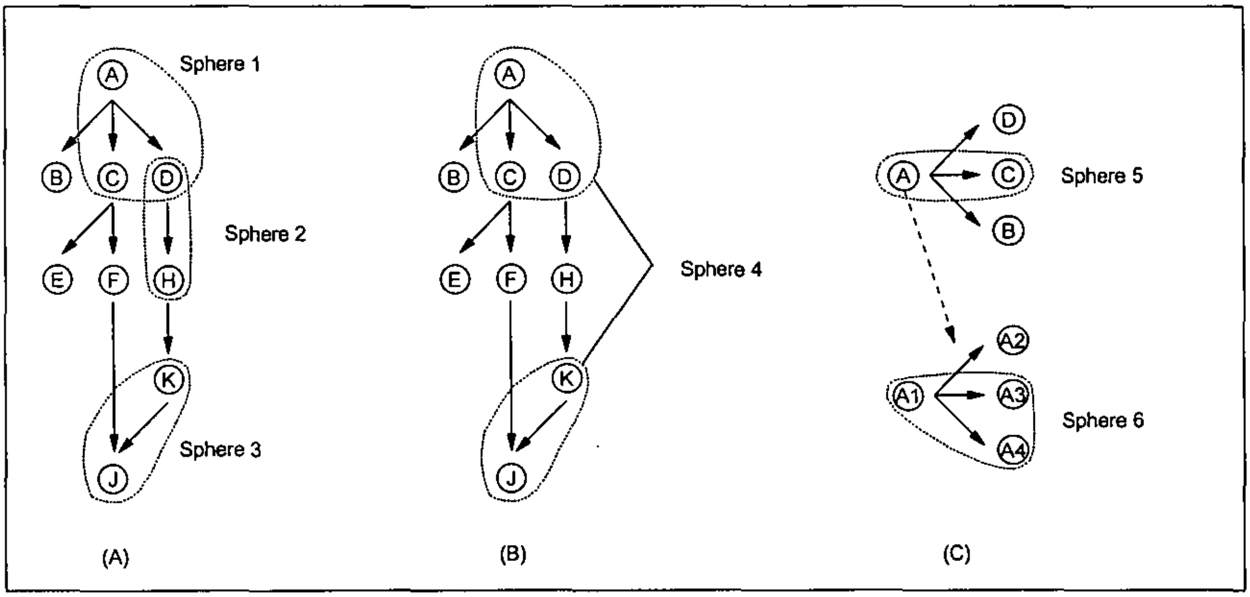


Figure 31: Example fragments of a Partial Rollbacks specification.

Consider for example process and sphere specifications from Figure 32. Assume that activities A, D, E, F, H, and J have been executed and that the execution of activity C fails. Compensation of previously executed activities is performed as follows. As the compensation type of the sphere 1 (to which activity C belongs) is set to discrete compensation (retry) the activities A and D are compensated in the reverse order of original execution. Thus first activity D is compensated for by executing -D and next activity A is compensated by executing -A. Assume here that activity A (as an initiating node) is rescheduled for execution. Furthermore as the proliferation property of sphere 1 is set to cascading and sphere 2 is dependent on sphere 1 with respect to activity D then sphere 2 has been affected as well. As the compensation type of sphere 2 is set to discrete compensation (undo) activities E and F must be compensated. Since activity F is a process itself either compensation activity –F can be executed (i.e. a shallow compensation is carried out) or the compensation activities contained in process 2 can be executed, i.e. activity –H, -I and –J (i.e. a deep compensation is carried out). In this case the nesting property of activity F is set to deep compensation, and therefore those activities executed as part of process 2 need to be compensated (i.e. by executing –H and –J).

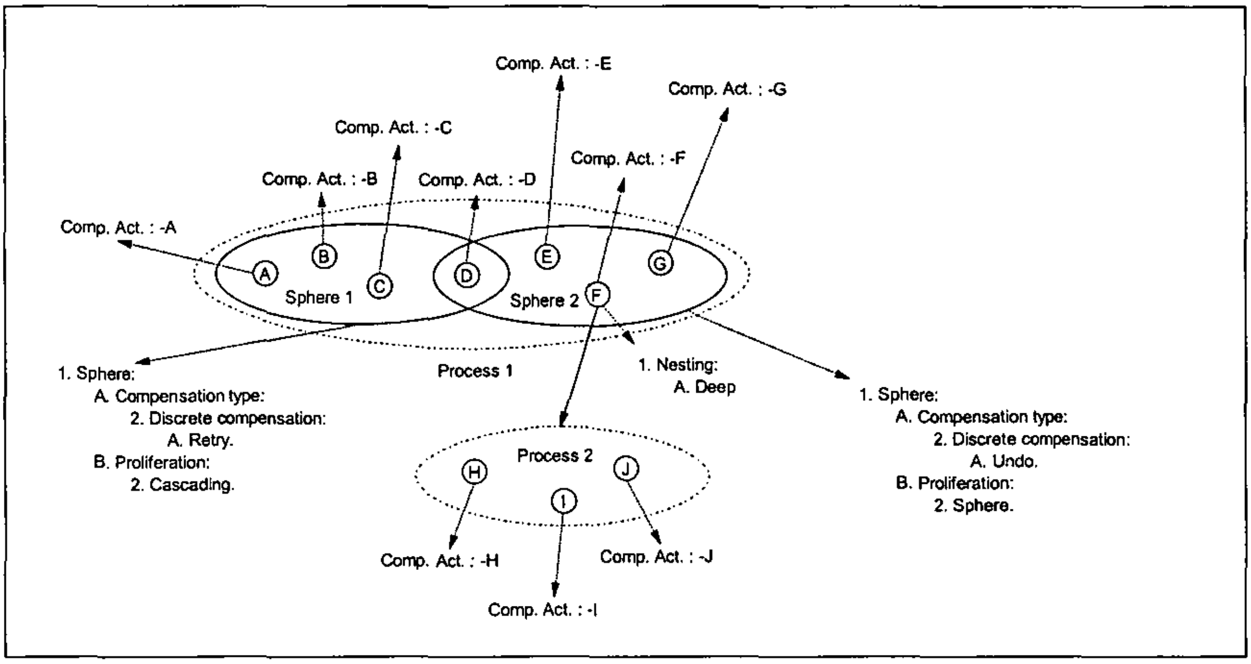


Figure 32: Example process and sphere specifications.

4.2 Summary

Properties of transactional workflow approaches can be considered from a number of alternative viewpoints as illustrated below.

1. Workflow Interdependencies: This viewpoint is concerned with how workflows become dependent on each other. The first of these workflow dependency types occurs where a sub workflow constitutes an activity in a super workflow. This kind of dependency is directly catered for by Sagas and Partial Rollbacks. An example of such a dependency is illustrated by Figure 33.A where sub workflow II constitutes activity C within the super workflow I. The second type of workflow dependency occurs where an activity belonging to one workflow becomes dependent on an activity in another workflow by nature of its use of data or flow of control. This kind of dependency can be handled by mechanisms provided by Partial Rollbacks. An example of this kind of dependency is illustrated in Figure 33.B, where activity B in workflow II is dependent on activity C in workflow I in terms of its use of data.

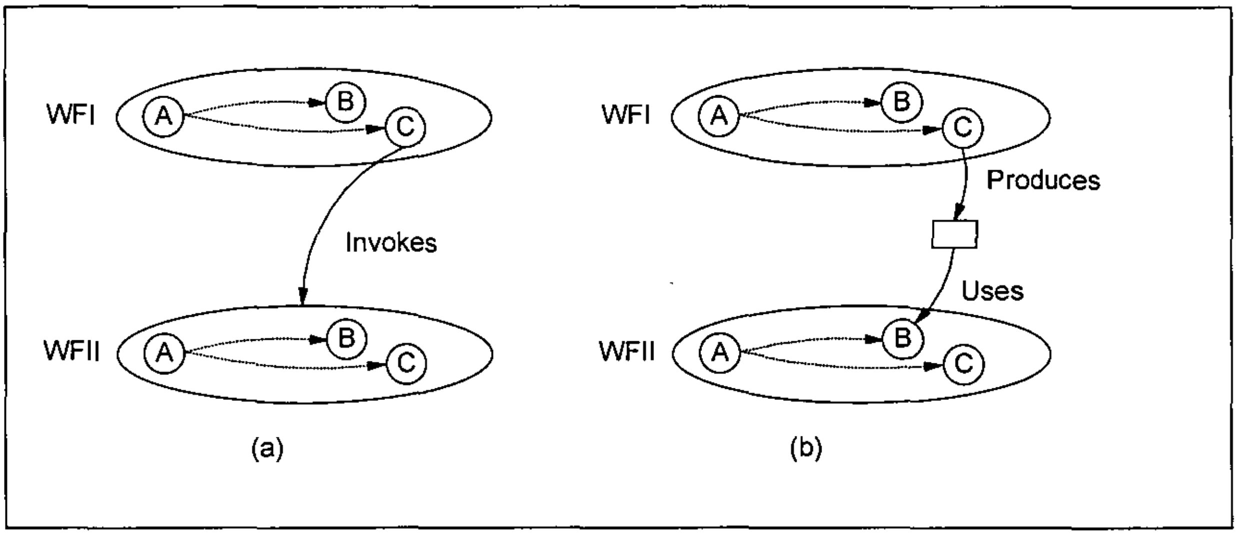


Figure 33: Workflow Interdependencies.

2. Scope of Compensation: this viewpoint is concerned with determining those activities in interdependent workflows that may need compensation as a consequence of a logical activity failure. Here three distinctive scopes of impact can be distinguished.

   I. For the first type, the scope is limited to a sub set of activities within a single and independent workflow in which a logical failure has occurred. The scope of the transactional workflow approach ConTracts is limited to a single workflow. An example of such a scope is illustrated by the workflow specification in Figure 34.A. If either activity B or activity C fails in its execution then the required compensation actions are limited to activities contained within this unit of work (in this example to activities B and C). The logical failure cannot impact on activities outside the scope of the unit of work (in the example the failure cannot impact on activities A and D).

   II. The second scope of compensation is limited to activities performed within interdependent super and sub workflows. A logical activity failure within a particular workflow may impact on the output of other activities from the same workflow which in turn can impact on the output of activities in super and sub workflows. Thus activities contained in sub workflows may be impacted on if their parent activity in a super workflow requires compensation. An activity in a super workflow can be impacted on if an activity contained in one of its sub workflow requires compensation. Sagas and Partial Rollback workflow specifications have underpinning mechanisms to compensate the output of activities contained in sub and super workflows. Consider for example a set of super and sub workflows, such as those illustrated by Figure 34.B. Assume that execution of activity B within workflow II has failed logically. Firstly compensation of any activity from super workflow I depends on the compensation properties of activity B from workflow I with respect to the susceptability of a logical failure of activity B from workflow II. Secondly any compensation of activities in sub workflow III depends on whether (1) activity C from workflow II will already have invoked sub workflow III and (2) compensation properties of activity C from workflow II with respect to the susceptability of this logical activity failure.

The third scope of compensation encompasses workflow types for which activities are linked by data and/or control flow dependencies. If activities in different workflows are linked by data and/or control flow dependencies, they may need compensation.

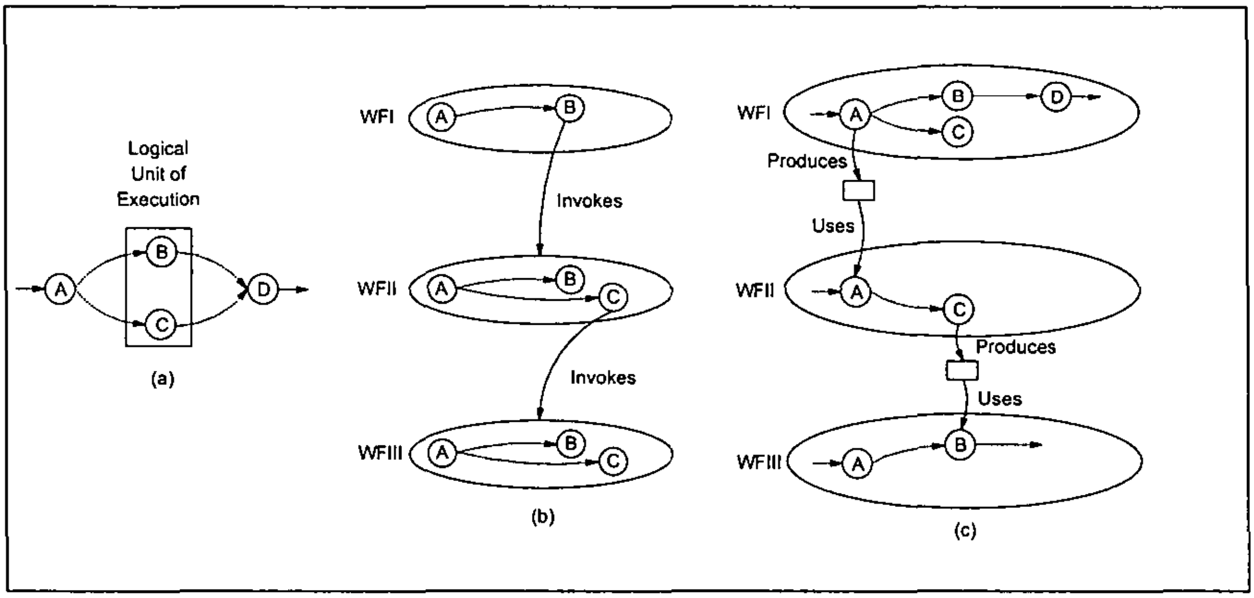


Figure 34: Scope of Compensation.

An example of this third type of scope of compensation is illustrated by Figure 34.C. Consider the scope of compensation for activity A in workflow I, where this activity is required to handle a logical activity failure of activity D in workflow I. Here the compensation scope will encompass: activities B, C and D contained in workflow I, where for this example there is a control flow dependency; activity A in workflow II, this involving a data flow dependency; activity C in workflow II, this giving rise to a control flow dependency with respect to activity A in workflow II; and activity C of workflow III, where there is a data flow dependency.

3. Type of compensation: This viewpoint is concerned with the extent to which the output of a previously performed activity must be undone from a semantic point of view before the activity can be re-performed. Two types of compensation may be distinguished namely: full and partial. Full compensation requires the output of an activity to be completely undone before the activity may be re-performed. Partial compensation only requires a part of the output of an activity to be undone before an activity may be restarted. Older transactional workflow approaches, such as Sagas and ConTracts, rely on full compensation of the output of previously performed activities, whereas the more recent workflow approach. Partial Rollbacks accepts that the output of previously performed activities may remain partly valid.
4. Compensation order: This viewpoint is concerned with the order in which the output of previously performed activities is compensated for, both within a workflow and across workflows. Compensation

can be performed (A) in parallel or (B) sequentially. Parallel compensation of the output of activities within a workflow does not itself impose any order on the start or completion of compensation activities with respect to each other. ConTracts provides mechanisms to handle parallel compensation actions within a workflow. Sequential compensation of the output of activities, however, can either be carried out (1) in the reverse order of execution to that of original workflow activities (such as by using mechanisms provided by Sagas and Partial Rollbacks), (2) in the same order as that for the original workflow activities (such as by using mechanisms provided by Partial Rollbacks) or (3) a combination of both.

Consider for example the workflow instance illustrated by Figure 35.A. This workflow consists of activities A, B, C and D. Parallel compensation of these activities means that compensation activities A-, B-, C- and D- will be started at the same instant in time, as illustrated by Figure 35.B. Sequential compensation actions for these activities in the reverse order to their execution are illustrated by Figure 35.C while compensation actions in the original order are illustrated by Figure 35.D.

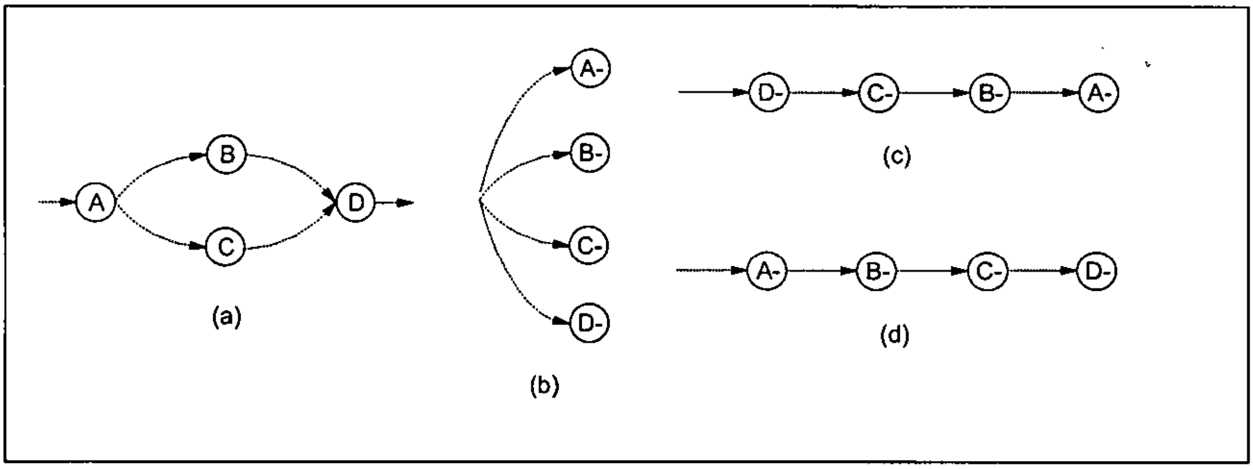


Figure 35: Compensation order.

Based on the preceding reasoning, Table 5 was constructed to compare the coverage of exception handling mechanisms provided by the transactional workflow approaches reviewed by the author.

	Relationships between Workflows	**Scope of Compensation**	**Full or Partial Compensation**	**Compensation Order**
Sagas	An activity from a super workflow can contain a sub workflow.	All previously executed activities in the workflow to which the failed activity belongs, all previously executed activities contained in invoked sub workflows (and their sub workflows and so on), all previously executed activities contained in any invoking super workflow (and its super workflow, and so on).	Full	(1) Compensation within a workflow is performed in the reverse order to that in which completed activities were performed. (2) Compensation across workflows is based on the following rules: (A) if an activity consists of a sub workflow then the output of all activities in this sub workflow must be compensated for before compensation may continue and (B) if the workflow has a super workflow (and all activities in the workflow) have been compensated for then activities in the super workflow must be compensated.
Partial Rollbacks	(1) An activity from a super workflow can contain a sub workflow or (2) an activity from a workflow can start a workflow at a peer level.	All previously executed activities in the workflow to which the failed activity belongs, all previously executed activities contained in invoked sub workflows (an their sub workflows and so on), all previously executed activities contained in any invoking super workflow (and its super workflow, and so on).	Full and Partial	Compensation action within a workflow can either be performed in the reverse order to that of the original activity flow or in the same order as the original activity flow.

Contracts	None. Workflow control is confined to within a single workflow.	All previously executed activities within a single unit of work that can be pre-specified as being part of the workflow specification.	Full	Compensation within a workflow is performed in parallel.

Table 5: Contrasting key capabilities of Sagas, Partial Rollbacks and ConTracts.

5.0 Discussion

From this review it can be concluded that a number of candidate technologies exist that might be used to specify and enact workflow approaches in application domains such as sales order processing. But further analysis and experiment is needed to understand how available technologies might fare with respect to 'static compensation' and 'dynamic compensation' requirements as defined and discussed in the following subsections.

5.1 Static compensation

Approaches to handling exceptions by means of static compensation are based on the concept that it is sufficient and practical to pre-define explicit sequences (or sets) of compensation activities for given exception types and their instances. They also assume that as exceptions occur during the useful lifetime of a workflow sequence it will prove practical to execute a predefined set of compensation activities to undo some of the previously performed standard activities[20] in order to return a workflow to a consistent state, from which point in time standard execution can continue. If the concept of static compensation is to be applied effectively, at design time a workflow designer must identify and specify standard activity flows and required compensation activities for each possible exception and workflow state that can arise when exception might impact. At execution time this 'cookbook' of compensation activities is simply executed.

Consider for example the workflow specification illustrated by Figure 36 which is encoded using IDEF3[21] traditional workflow modelling constructs. Here the standard workflow specification part defines the order in which activities A, B, C and D should be performed whilst the exception handling specification part defines explicit sequences of compensation activity needed to handle exceptions FAIL1, FAIL2 and FAIL3 that may occur respectively at activity C and D. If for example exception FAIL3 occurs at activity D compensation activity $A^{-\frac{1}{2}}$ must be executed followed by compensation activity $C^{-\frac{1}{2}}$. Subsequently standard workflow execution can continue from standard activity D.

---

[20] The term 'standard activities' is used to imply an activity unit which is part of a predefined 'standard process' that will be followed unless exceptions occur.

[21] As described in section 2.1.5 of chapter 4.

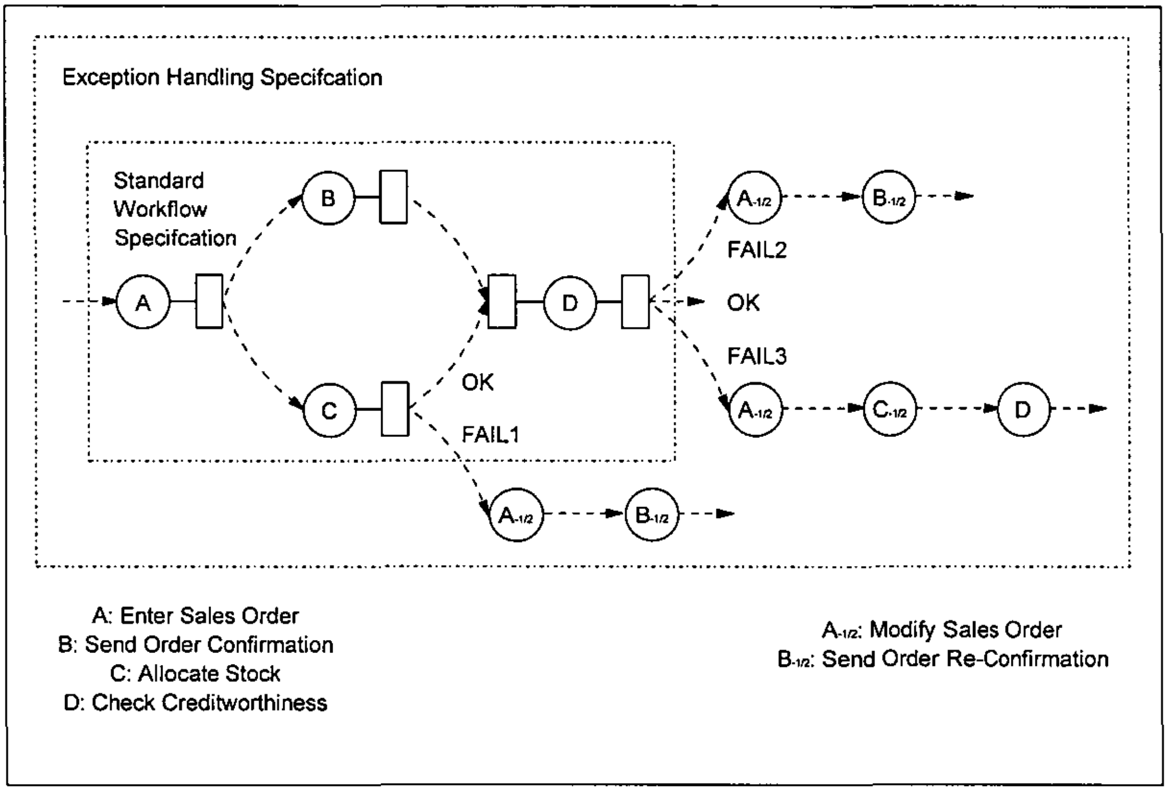

Exception Handling Specifcation
Standard Workflow Specifcation
A
B
C
D
OK
FAIL1
FAIL2
FAIL3
$A_{-1/2}$
$B_{-1/2}$
$C_{-1/2}$
A: Enter Sales Order
B: Send Order Confirmation
C: Allocate Stock
D: Check Creditworthiness
$A_{-1/2}$: Modify Sales Order
$B_{-1/2}$: Send Order Re-Confirmation

Figure 36: Workflow specification with explicit sequences of compensation activities.

5.2 Dynamic compensation

Approaches to handling exceptions by means of dynamic compensation are founded on the idea that necessary compensation activities can be identified and specification generated dynamically as the workflow execution is ongoing. This may necessitate use of some exception handling specification, knowledge about the state of the workflow and some general rules. Although some form of exception handling specification may be needed they may not need to consist of an explicit definition of compensation activities that must be executed in the case of a given exception. On the contrary they might capture some form of 'high-level knowledge' that can be used to handle exceptions. Indeed a workflow designer might define certain kinds of exception handling specification and general rules that can be used as input to computational processes that specifically predict necessary compensation activities for a given exception and workflow state, possibly by making reference to characteristic properties of the domain in which they are currently being used. If appropriate types of specification and rule can be defined they might be reused with relative ease in support of the analysis and specification of standard workflow and exception handling specifications in various manufacturing domains.

Consider for example the workflow specification illustrated by Figure 37.A and B which is based on use of ConTracts[22] transactional workflow modelling concepts and constructs. The standard workflow specification part illustrated by Figure 37.A defines the order in which standard activities A, B, C and D need to be executed whilst the exception handling specification part illustrated by Figure 37.B defines those activities A, B and C form an 'atomic unit of work'.

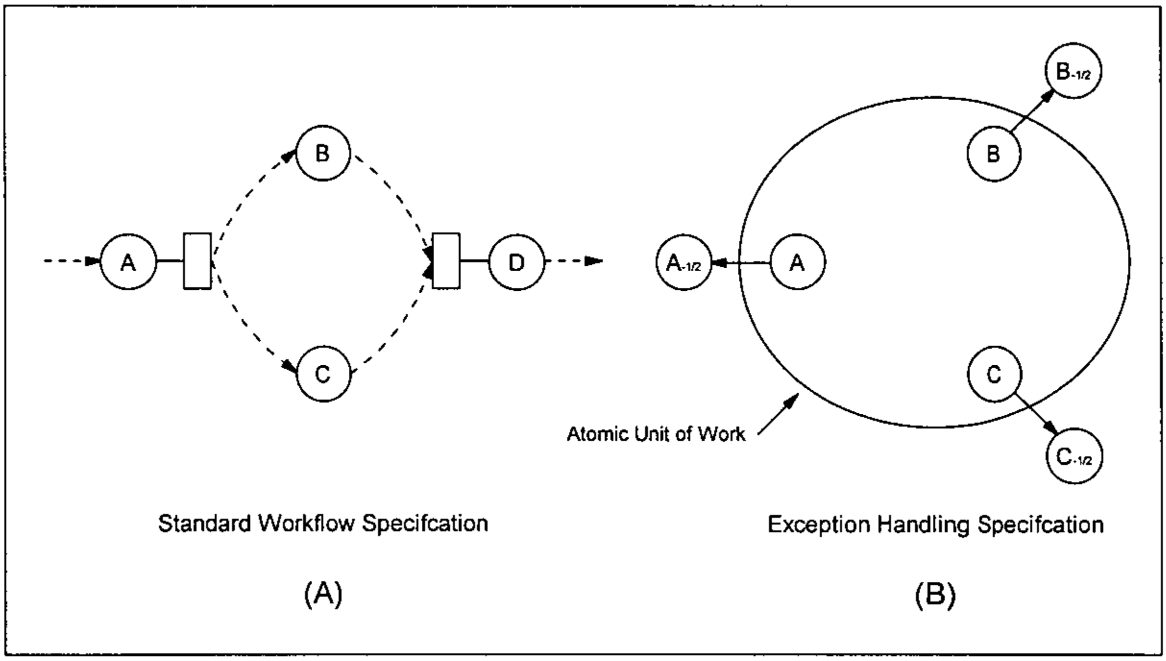


Figure 37: Exception handling specification without explicit compensation activities.

The rules that underpin this design specification and enactment of this workflow approach are as follows. If an activity completes and no exception occurs, other activities may be started as defined by the standard workflow. However if an exception occurs, and the failed activity belongs to a given 'atomic unit of work' as specified by the exception handling specification, those activities that have been performed within the attached work unit need to be compensated in their original order of execution. If therefore the workflow state indicates that these activities have already been executed then appropriate compensation activities should be generated dynamically. The said compensation activities can be determined with reference to suitable predefined exception handling specifications.

Thus if either an exception occurs at activity A, B or C then activities A, B and C may need to be compensated depending on whether they have been performed or not. Consider for example four of the states that this workflow may be in, namely: (1) activity A may have been performed, (2) activities A and B may have been performed, (3) activities A and C may have been performed, or (4) activities A, B and C may have been performed (as illustrated in Figure 38). Assume that the

---

[22] As described in section 4.1.2 of chapter 4.

execution of activity B fails. In the case of workflow state (2) compensation activity $A^{-1/2}$ is executed followed by compensation activity $B^{-1/2}$. However in the case of state (4) compensation activities $A^{-1/2}$, $B^{-1/2}$ and $C^{-1/2}$ are generated. In this way different sets of compensation activities can be generated depending on the state of a workflow.

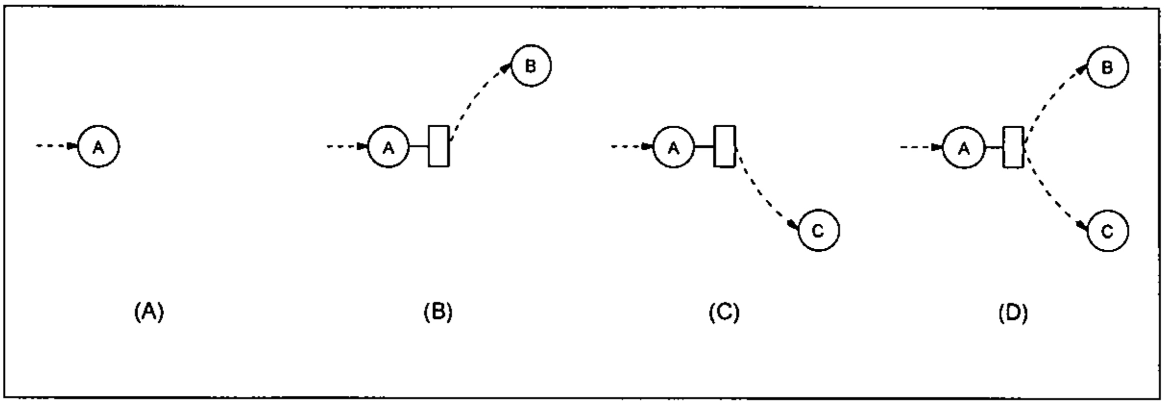


Figure 38: Possible workflow states when an exception might occur.

5. A new workflow model for sales order processing

1.0 Introduction

This chapter will analyse and define essential characteristics of exception handling within the domain of sales order processing and will formulate a set of general end-user and workflow system developer requirements that workflow approaches should meet to function effectively in that domain.

2.0 Sales order data and control flows

During the last two decades many kinds of workflow approaches have been proposed by research that have resulted in new and important concepts, such as those reviewed in chapter 4. However some new techniques and mechanisms have been proposed whilst the rationale behind these approaches has been given relatively little attention. Therefore before any new approach was developed as part of this study, it was considered to be important to understand in detail the nature of domain requirements. Prior to this study little knowledge was available about the types of domain exceptions that occur, about their properties and effects on the state of a workflow or about how exceptions propagate. This study observed that many exceptions in sales order processing can arise because of change in customer order quantity and due dates, stock allocation problems, customer creditworthiness problems, etc. But it needed to be understood whether these frequently occurring domain exceptions were similar in nature or different before choosing suitable workflow exception handling concepts and mechanisms. It was decided that such an analysis needed to be carried out with respect to a particular application domain because from practical experience it was argued that it is difficult to make statements about exceptions and their effects that hold in every case. Thus the scope of this study and its associated new workflow approach design and development was limited to the domain of sales order processing, albeit that the intention was that ultimately the study would also consider similar domains of application like credit loan applications. Therefore it was hypothesised that once more specific issues are better understood it may prove possible to reason about more general issues in handling exceptions and to derive a set of requirements that a workflow approach should meet from a broader set of sales order processor and workflow developer perspectives.

2.1 Sales order data flows

This section describes how sales order processing activities are generally interrelated, the kind of exceptions that may occur in sales order application domains and their likely effect on the output of sales order processing activities, and how the effect of exceptions can propagate to cause impact on other interrelated activities.

Key aspects of sales order activities can be represented within a computer by encoding how such an activity receives input data from the real system (this being via 'external inputs') and from upstream activities (via 'internal inputs' generated by other activity elements of the computer model of sales order processing). Having operated on their inputs, in general sales order processing activities will generate data outputs that impact on the real system (via external outputs) and become inputs to other downstream activities (via internal outputs). This is illustrated by Figure 39. Generally activities will also have associated post-conditions that indicate whether or not execution of an activity has been successful.

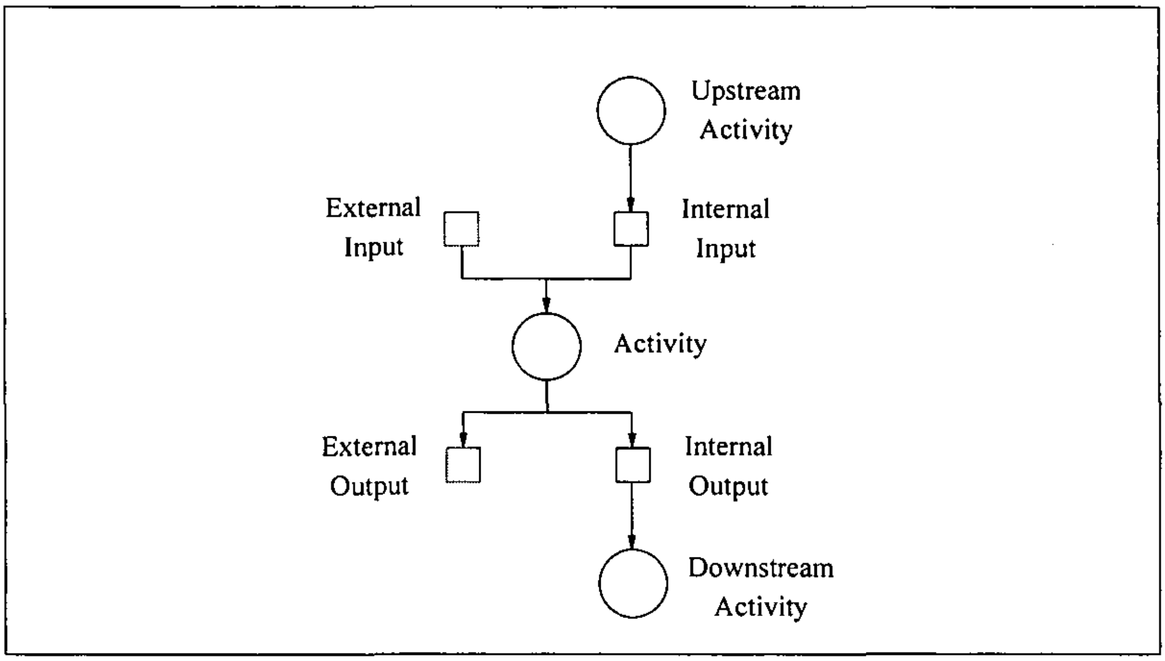


Figure 39: Data input and data output of an activity.

Thus we observe that output of any upstream sales order processing activity can be used as an input to many downstream activities. Also downstream activities may use the output of a number of upstream activities as their input (as illustrated by Figure 40). In this way a network of dependent activities is generated.

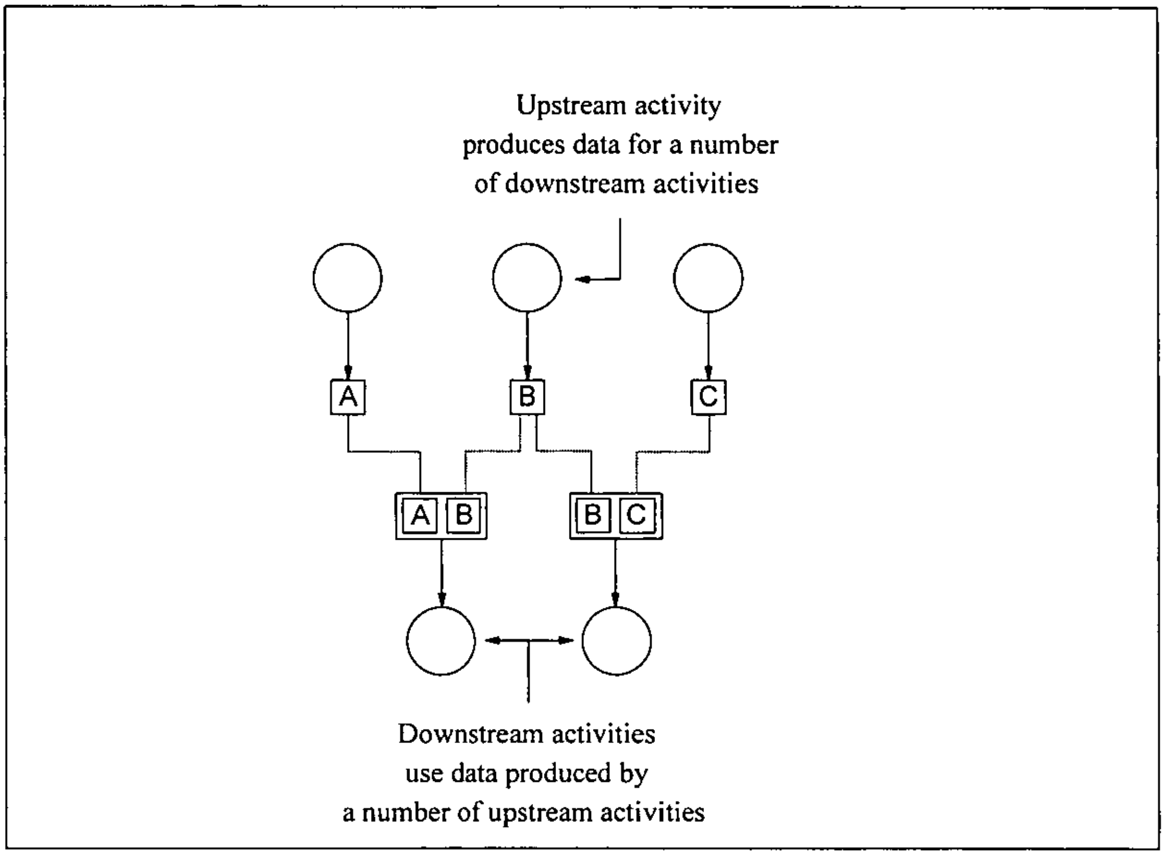


Figure 40: Data flows between up- and downstream activities.

Consider the example of a sales order data flow comprising: activity A 'Enter Sales Order'; activity B 'Allocate Stock'; activity C 'Send Order Confirmation'; and a master production schedule data flow that consists of activity D 'Enter MPS (Master Production Schedule)'. Figure 41 illustrates such an example sales order and master production schedule data flow. In this case the activity 'Enter Sales Order' takes a sales order as defined by some external input from a customer and generates a sales order in the form required as an internal output. This sales order is used as an internal data input by the activity 'Send Order Confirmation', that in turn generates a paper-based or electronic confirmation as an external data output. In a similar way the activity 'Enter MPS' takes in external MPS data input and simply stores its results as an internal output. The activity 'Allocate Stock' uses as internal inputs a sales order and a master production schedule to generate a list of allocated stock items as an internal output.

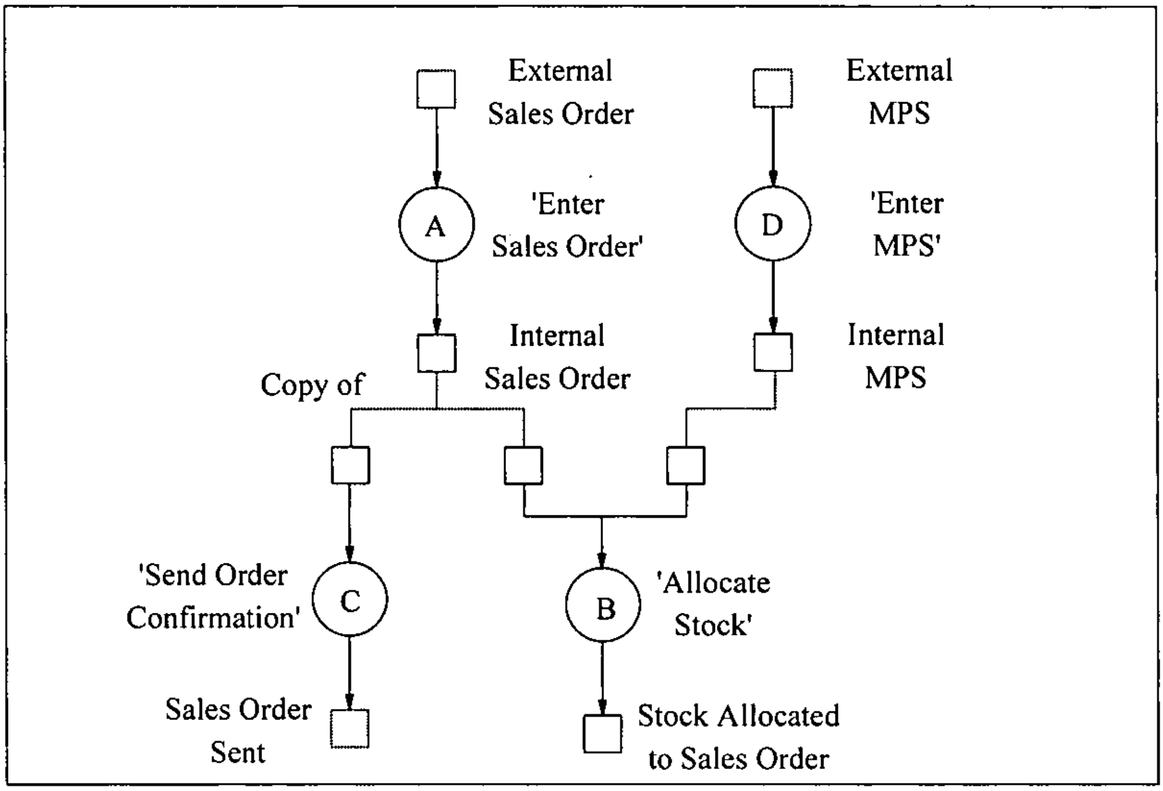


Figure 41: Example sales order and master production schedule data flows.

2.1.1 Change to the input of a completed activity

Sales order processing exceptions can arise as a consequence of a change to the input of a completed activity. This can occur if either (A) the real system makes a change to external data provided as an input to an activity (this being considered in this study to be a root exception) or (B) if an upstream activity makes a change to its output and this is relayed as a change to the input of a downstream activity (this being considered in this study to be a derived exception). Figure 42.A illustrates an occurrence of both root and derived exceptions. In such situations the new input of an activity is not consistent with its original output, according to the logic which describes the operation of the activity. This point is illustrated by Figure 42.B. However when the output is updated it may already be the case that the original output had been used by a number of downstream activities (within the same workflow or across workflows) and such a condition should trigger derived exceptions, as illustrated by Figure 42.C.

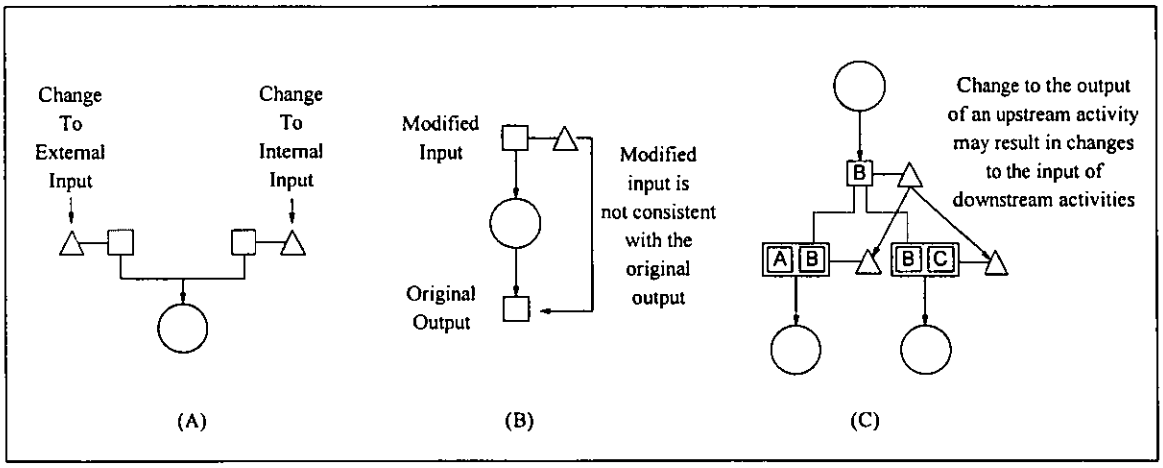


Figure 42: Illustrative situations following a change to the input of a completed activity.

Consider again the sales order data flow example illustrated by Figure 41. Suppose the customer wishes to reduce the number of products ordered by 30. This constitutes a change originated by the real system to the external input of the activity 'Enter Sales Order' (see Figure 43). Under such conditions the output of this activity is no longer consistent with the new input data and changes need to be made to update the output, as illustrated by Figure 43. If the original output of this activity has already been used as an input by the downstream activities 'Send Order Confirmation' and 'Allocate Stock' the change in output of the 'Enter Sales Order' activity also needs to be propagated to their inputs (this is also illustrated by Figure 43).

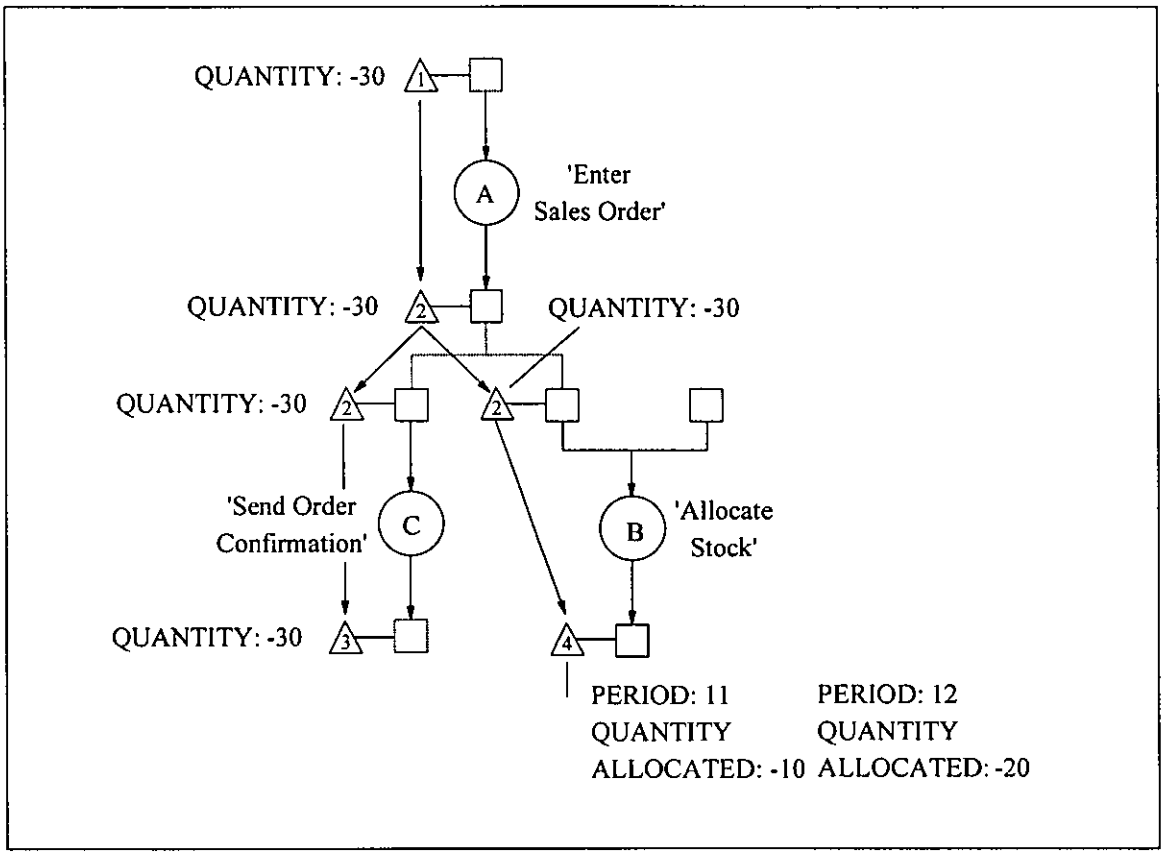


Figure 43: Illustrative effect of a change to the input data of the completed activity 'Enter Sales Order'.

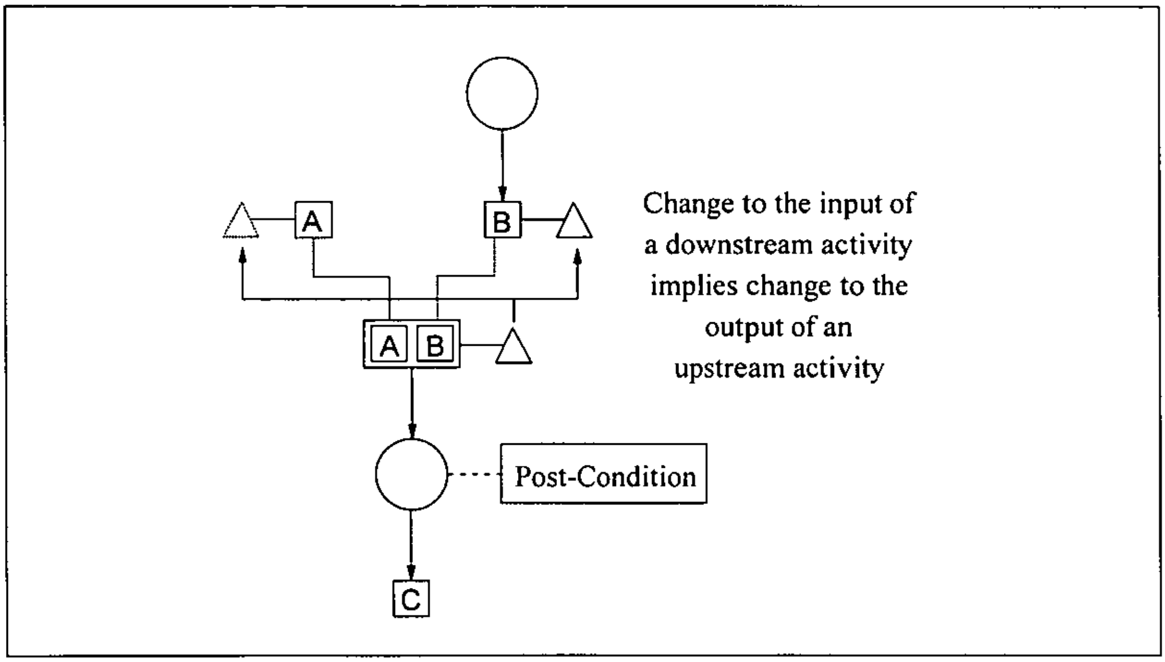
2.1.2 Logical activity failure

An exception corresponding to the logical failure of a sales order processing activity occurs when an activity completes its execution but its associated post conditions cannot be met. Under such conditions it proves impossible to use the logic of an activity to transform a given input into such an output that the activity's post-conditions are met. Therefore the input of the activity must be changed if the required activity transformation is to be successful the next time it is executed[23]. But this implies that the output of an upstream activity (or of multiple upstream activities) or the data provided by the real system must also change since the input of the activity must have originated from somewhere. Such a situation is illustrated by Figure 44. This further implies that (a) original input data to an upstream activity (or activities) may not correspond to new (or proposed) activity outputs according to the logic of the activity concerned or (b) that the real system providing input data should consider or reconsider the impact of the proposed changes. If

---

[23] Here we assume that the logic of an activity is deterministic, i.e. for every input there is a unique output.

case (a) holds true then the input to an activity must change in order for it to be consistent with its new output, but this in itself constitutes another exception.



Change to the input of a downstream activity implies change to the output of an upstream activity

Figure 44: Effect of a logical activity failure.

Consider once more the sales order data flow and the master production schedule data flow examples illustrated by Figure 41. The activity 'Allocate Stock' has an associated post-condition. This dictates that the quantity of products allocated from a MPS to a sales order must equal the number of products ordered by the customer. Assume that the activity 'Allocate Stock' fails in its execution because in practice it is not possible to allocate 20 components by the required due date. Assume in the example workflow that the input data (100 components before week 11) is converted (unsuccessfully) into the following outputs: 50 components in week 9 and 30 components in week 10. It follows that the input data has to be changed in one of the following ways. (1) Available quantities in the MPS can be increased by 20, as illustrated in Figure 45.A. (2) The required quantity is decreased by 20, as illustrated in Figure 45.B. (3) The required due date is delayed by 1 week. (4) Some viable combination of all these options is deployed.

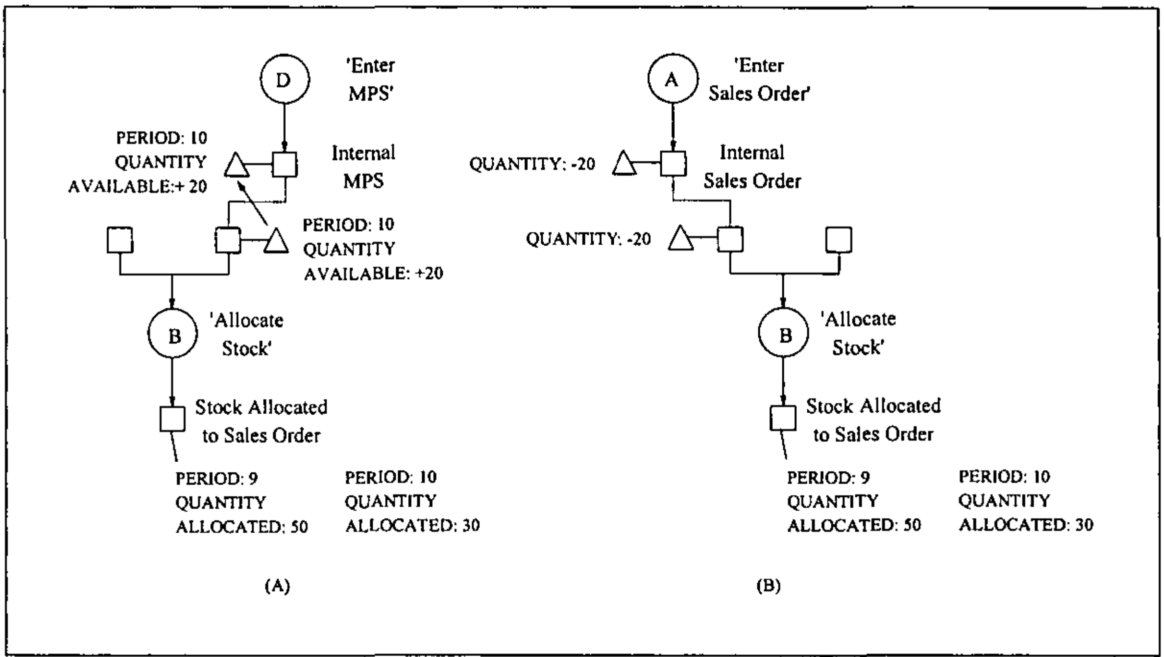

D
'Enter MPS'
PERIOD: 10 QUANTITY AVAILABLE:+ 20
Internal MPS
PERIOD: 10 QUANTITY AVAILABLE: +20
B
'Allocate Stock'
Stock Allocated to Sales Order
PERIOD: 9 QUANTITY ALLOCATED: 50
PERIOD: 10 QUANTITY ALLOCATED: 30
(A)
A
'Enter Sales Order'
QUANTITY: -20
Internal Sales Order
QUANTITY: -20
B
'Allocate Stock'
Stock Allocated to Sales Order
PERIOD: 9 QUANTITY ALLOCATED: 50
PERIOD: 10 QUANTITY ALLOCATED: 30
(B)

Figure 45: Effect on data dependent activities for a logical failure of activity 'Allocate Stock'.

2.2 Sales order control flows

Although the flow of data already imposes an order on the way in which activities should be executed[24], it is generally the case that manufacturing companies deploy sets of business rules to impose further order on the execution of sales order activities. Such business rules are capable of capturing specific practice, e.g. to handle recurring situations in a company in a cost effective and prompt way. These rules determine what appropriate activities should be performed in a particular situation at a particular time. As such they should prevent the user from executing activities that are not required or from executing activities that are required but should not occur at the wrong time.

Use of a generalised business rule is proposed in this research study which orders the execution of sales order-activities in terms of a conditional sequence. Although there are other generalised business rules (such as temporal rules [Ver96, Luc95a]) that are applied to order the execution of sales order-activities, use of these other generalised business rules was left outside the scope of this research study so that its objectives were not overly ambitious. Therefore it has not been an aim to study different ways of ordering the execution of sales order-activities, but emphasis has

---

[24] In the data flow model a consumer activity may only be executed after the producer activity has been executed.

been placed on defining and enabling the effect of exceptions on the order in which activities are executed in the sales order domain. As the 'conditional sequence' business rule can capture the logic of many practical situations it was observed that it would be important to study in detail issues of handling exceptions that are associated with use of this business rule before broadening the study to include other kinds of business rule.

Therefore the generalised business rule studied in this research is based on the assumption that an activity may be started after (A) the completion of a particular set of activities and (B) the values of output variables of previously performed activities have met particular conditions. Part (A) of this rule can be expressed in text form as follows: activity set → activity [operator], where (a) an activity set may comprise either a single activity or multiple activities and (b) an operator may be AND, XOR or OR. Part (B) of this rule can be expressed in the text form: (output variable activity; operator; constant)* where an operator is =, >, <, =>, or =<. (* indicates that there may be 0, 1 or many of these parts).[25]

For example consider a data flow specification comprising a producer activity A and consumer activities B, C, D and E[26], as illustrated by Figure 46.A. Activity A may only be performed if start event S occurs (rule S → A). Activity B may only be performed if activity A has previously been performed and the value of the output variable Q of activity A is larger than 100 (rule A → B; A.Q > 100). Further activity C may only be performed when activity A has been performed and the value of the output variable Q of activity A is smaller or equal to 100 (rule A → B; A.Q <= 100). Whereas activity D may only be performed when activity B or C has been completed (rule (B OR C) → D) and activity E should not to be performed for this company at all.

---

[25] Here it is conceptualised that a special signal external to the control flow has occurred, i.e. a start event. This rule can be expressed in the text form S → activity.

[26] Where activity A may be 'Enter Sales Order', activity B may be 'Generate Purchase Order', activity C may be 'Generate Works Order', activity D may be 'Send Order Confirmation' and activity E may be 'Allocate Stock'.

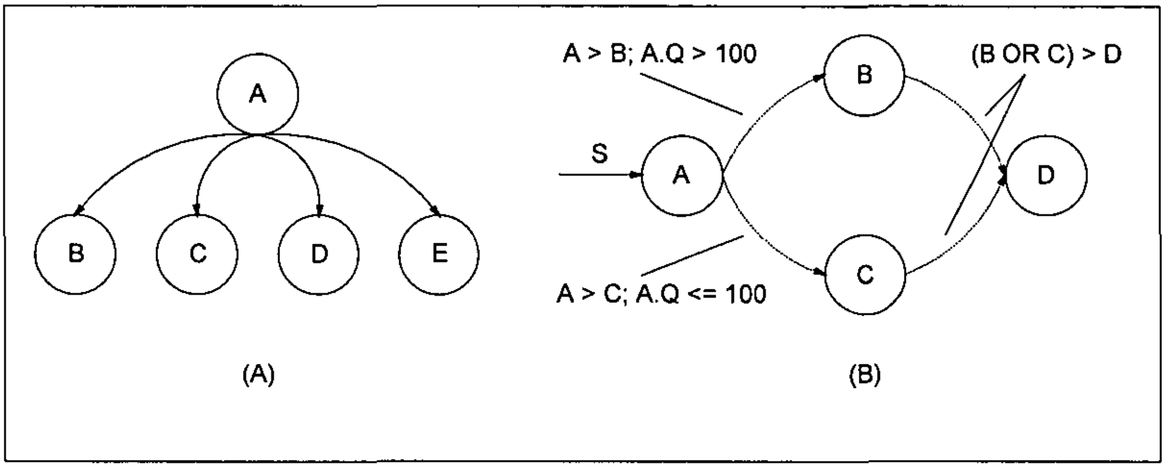


Figure 46: A data flow and a control flow specification.

2.2.1 Cancel event

A cancel event has been considered in this study to correspond to the occurrence of an external event that indicates that there is no need anymore to perform the activity associated with some start event, nor is it necessary to perform other activities that have been enabled by business rules related to the activity, and so on. Thus a direct effect of a cancel event is that the activity enabled by a start event should be cancelled (i.e. the output of this activity should be discarded). Also an indirect effect of a cancel event is that the application of a number of business rules associated with an activity may no longer be valid and hence that other activities enabled by these business rules should be cancelled.

Consider again the control flow specification illustrated by Figure 46.B and the control flow instance based on this control flow specification which consists of activities A and B, as illustrated by Figure 47.A. Assume that activity A has been executed after an external start event has occurred. Assume also that activity B has been executed since it was enabled for execution because of the application of business rule A → B. Assume at this point in time a cancel event occurs. This situation requires the output of activity A to be cancelled, as illustrated by Figure 47.B. Further business rule A → B is no longer valid since activity A has now been cancelled.

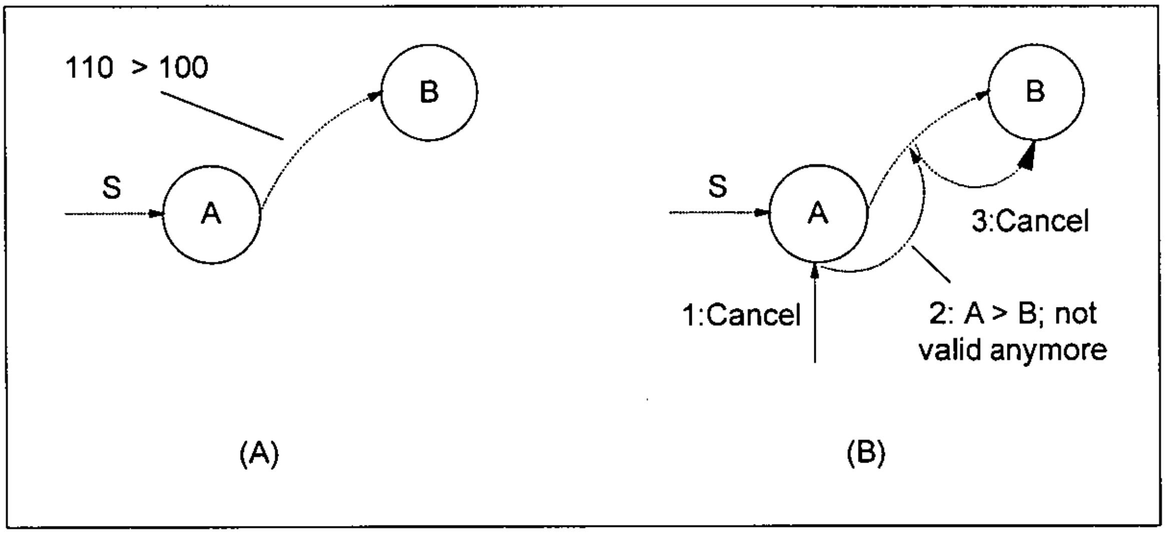


Figure 47: Example control flow specification and control flow instance.

2.2.2 Effect of a change to the input of a completed activity on the status of business rules

We have seen in section 2.1 of this chapter that one class of sales order processing exception occurs when there is an input change to an activity that has previously completed. As a change to the input of a completed activity will generally require a change to the output of the activity and because business rules may be applied based on the state or value of that output then the states of decision logic associated with these business rules may also change. For example the state of a business rule may be:

1. originally true, but become false after the exception has occurred. This implies that an activity triggered by this business rule should not have been performed in the first place and ideally should be cancelled. Further other business rules that are based on the results of this activity may also have become invalid. This implies that in general activities associated with the application of such business rules may also need to be cancelled, i.e. a derived exception has occurred. As a consequence a path of activities may need to be cancelled.
2. originally false, but become true after the exception has occurred. This means that the activity associated with such a business rule is now enabled for execution.

Consider for example the control flow specification consisting of activities A, B, C, D and E[27], and a control flow instance based on this control flow specification that consists of activities A, B and C (as illustrated by Figure 48).

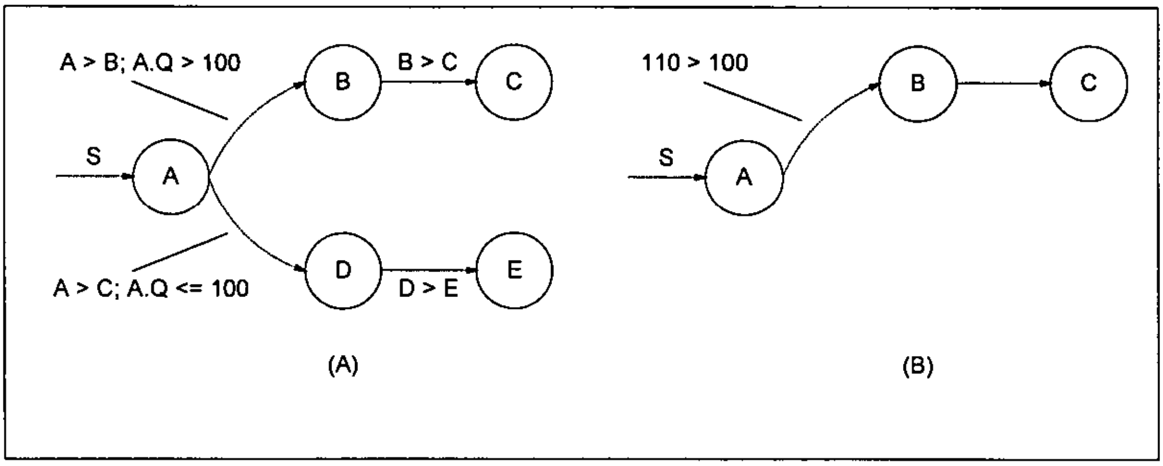


Figure 48: Control flow specification and control flow instance.

Assume that an exception corresponding to a change to the input of completed activity A occurs and that this exception results in a change to the output of activity A (e.g. Q = 110 changes to Q = 90), as illustrated by Figure 49. Under such conditions the business rule A.Q > 100 is no longer valid (since 90 > 100 is not true) and therefore activity B should not have been performed and ideally its output should be cancelled (i.e. a derived exception occurs). As a consequence business rule B > C is no longer valid either (as activity B has been cancelled) and therefore activity C must also be cancelled (yet another derived exception occurs). On the other hand business rule A > C; A.Q <= 100 is now enabled and therefore the execution of activity D may be started.

---

[27] Where activity A may be 'Enter Sales Order', activity B may be 'Generate Purchase Order', activity C may be 'Release Purchase Order', activity D may be 'Allocate Stock' and activity E may be 'Generate Works Order'.

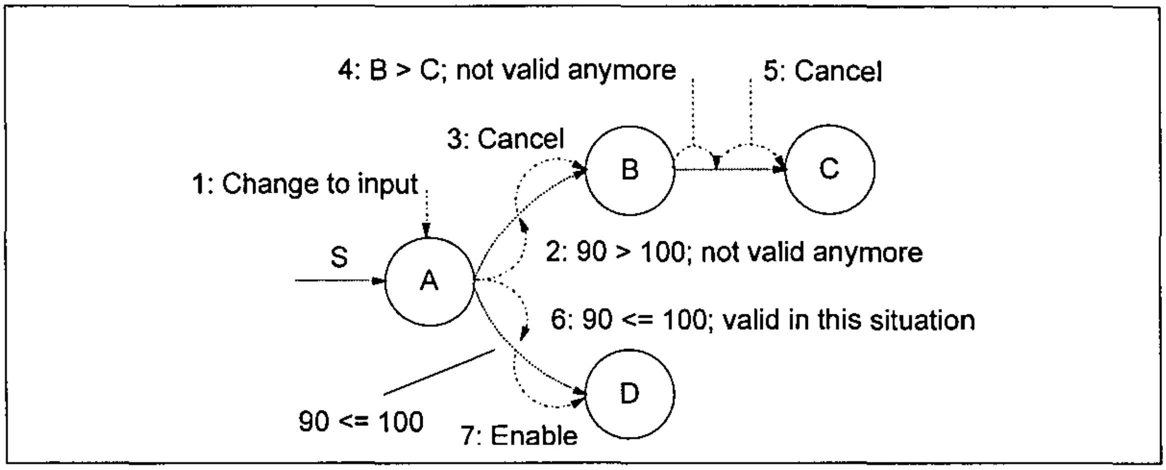

4: B > C; not valid anymore
5: Cancel
3: Cancel
1: Change to input
B
C
S
A
2: 90 > 100; not valid anymore
6: 90 <= 100; valid in this situation
D
90 <= 100
7: Enable

Figure 49: Effect of an exception corresponding to a change to the input of a completed activity.

2.3 Discussion

The results of this study into the properties and effects of exceptions within the context of sales order processing can be summarised as follows:

(1) Three types of exceptions have been observed and characterised that can represent many exceptions that may occur in the domain of sales order processing, namely input changes, logical failures and business rule changes.

(2) Two kinds of dependencies between activities in the sales order processing domain have been observed and characterised namely, i.e. data and control flow dependencies, and their role in propagating the effect of exceptions. Two kinds of effects on control flows were recognised: (1) paths of execution that turn invalid and (2) new paths of execution that are entered. Furthermore upstream activities may require downstream activities to be redone (and vice-versa) because of their data-dependencies. There may also be interaction between these flows, constituting an exception to a data flow that may result in an exception to a flow of control.

The value of this research is as follows. First of all this set of domain observations can characterise many exceptions that actually occur within the sales order processing domain and can possibly form the basis of a 'domain theory' which can be used to identify exceptions and determine suitable exception responses in a systematic way. This might for example enable fewer exceptions to go undetected at the time of design and first appear at the time of execution when they will need to be solved at a greater expense. Secondly such a theory should facilitate a determination of the effects of exceptions on activities already performed in a sales order workflow. It should provide a basis for reasoning about all necessary compensation activities needed to handle such an exception.

Yet within the context of the proposed model for sales order processing other kinds of exceptions may occur that impact on the state of a workflow instance. Firstly it may happen that the post-conditions of an activity change and this may require activities to be redone. For example a condition may be added to the activity 'allocate stock' that dictates that all stock allocated to an order must be in a window of two weeks before the due date; thus orders already planned but outside this window should be adjusted. Secondly the activity logic may change and if this logic was originally used different outputs would be produced. Neither of these kinds of exception are addressed in this study. Yet it is likely that these kinds of exceptions will occur less frequently than the three kinds of exceptions discussed earlier in this section. Finally if the logic of the activities can be modelled it should be possible to determine exactly how exceptions filter through activity networks[28].

Therefore, in essence, the author of this research study proposes that data and control flows really bind sales order processing activities. However it is important to note that there may be other paradigm aspects of modelling exceptions and their effects. For example it may be sufficient to model important plans such as delivery schedules, stock allocation plans, etc. and their relationships to exceptions. Yet inevitably other aspects will be left out such as for example any business rules that must be taken into account. Indeed the effects of exceptions such as activity input changes and logical activity failures are naturally explained if data flows are taken into account. Indeed data and control flow modelling are widely established by models such as SADT/IDEF0, CIM-OSA, IEM, Grai Nets and IDEF3. Also many issues of workflow modelling have not been addressed like organisational issues (authority & responsibility), resources, etc. which are traditionally considered in the context of enterprise modelling.

---

[28] Here it is also to be noted that activities have to convert input to output in the same way; however this can be the case as the activity logic is well-known.

3.0 General competences of workflow approaches

This section of the thesis identifies three complementary ways of qualifying general 'competences' of workflow approaches. Two main groups of user of these competence measures were envisaged, namely (1) those persons concerned with a specific workflow application and (2) those persons concerned with developing and appraising relativities between different types and generations of workflow approaches. The first target group of 'users of the competence measures' will be termed 'end users of workflow approaches', whilst the second target group will be termed 'developers of workflow approaches'. It may be instructive to point out that the author, his supervisor, external examiner and other developer members of the workflow community are naturally members of the second target group. During this study the author also played various user roles within the first target group, as he applied the new workflow approach in the Sales Order Processing (SOP) domain.

During the study it became evident that an 'absolute' completeness measure was required to quantify any set of functional capabilities provided by a workflow approach. The degree of completeness should be determined by making reference to the complete set of workflow approach functionality requirements found in the application domain in which the approach is to be deployed. Naturally therefore any value attributed to the completeness criterion will be context dependent. End users of a workflow approach could beneficially utilise completeness criteria, when judging the quality (i.e. fitness for purpose) of an approach. It also became evident during this study that 'effort & flexibility' and 'correctness' workflow approach competency measures were required to enable end users and developers of workflow systems to make quantitative judgements about the 'rate at which' and 'ease with which' any given workflow approach can be created and used during its lifetime. Thus the development of a workflow approach is not only complicated by the issue of completeness, but by the issue of effort & flexibility, and the issue of correctness.

Therefore the prime focus of this study has been to design flexibility into a new workflow approach. In the SOP domain a capability to develop and utilise simple and standard flow definitions yet be able to handle prime types of exception has been seen as offering significant potential to improve upon both the utility and flexibility of current generation workflow systems. From a flexibility viewpoint such a capability would enable a significant improvement in (1) 'workflow reactivity' and (2) 'workflow programmability'. Here it is assumed that (1) improved reactivity, i.e. a capability to respond to uncertain end-user exceptions, and (2) improved programmability, i.e. a capability to respond to uncertain system developer exceptions (as defined by Weston 1999) will arise through providing formal and structured means of handling exceptions in a manner which is largely de-coupled from any method used to program standard workflows. Linked directly to this is a key proposition developed by this study that a problem decomposition, based on separating standard workflows from exception workflows, should improve the scalability of workflow solutions. In turn it is supposed that if separate and effective means of handling standard and exception workflows can be provided then the effort required to facilitate 'programmability' and 'reactivity' should be much reduced. Consequently this approach to workflow decomposition approach should lead to more flexible workflow systems, in that they can practically (A) reach an increased number of possible states and/or (B) permit a broader spectrum of requirements change.

The following subsections explain how in this study general completeness, effort & flexibility and correctness requirements were identified and used to develop workflow approach competence measures that enabled the author to (a) draw out observations and generalisations about the established literature on workflows and (b) guide the architectural design and testing of a new and improved workflow approach.

3.1 Completeness competence

Section 2 of this chapter defined the different effects of various kinds of exceptions to a sales order workflow on previously performed activities. Based on these observations it was found that a workflow approach for sales order processing must be capable of:

1. Capturing the exception types of (A) a logical activity failure, (B) a change to the input/output of a completed activity and (C) a change to a state of a business rule. If a workflow approach is not capable of capturing occurrences of these types of exception, even though it may provide exception handling techniques, it cannot detect important exceptions that may be common in any sales order processing application.

2. Generating a sequence of compensation activities for any type of exception and any workflow state at execution time from some exception handling specification defined by the workflow designer at design time. Ideally a workflow approach for sales order processing should come with rules to identify which activities and how activities in a workflow have been affected by an exception:

    I. To identify those business rules that are not valid anymore and those business rules that were originally not valid but are now valid after the occurrence of an exception. Business rules that are not valid anymore point to completed activities in a workflow that needs to be cancelled. Thus invalid business rules and activities to be cancelled constitute invalid paths of execution. Business rules that become valid after the occurrence of an exception point to new paths of execution that may be started if an exception is accepted.

    II. To identify those completed activities in a workflow for which an input or output has changed and which need to be updated. Activities to be updated can be found on still valid paths of execution.

III. To identify those activities in a workflow that may be started by the end user anytime but for which their output is uncertain or may not be required anymore, and thus needs to be blocked. Activities to be blocked can be found on still valid paths of execution.

An example of this requirement is given in the paragraph below, namely requirement (3).

3. Determining whether an exception can be handled or not, before executing the proposed sequence of compensation activities. If either a static pre-specified sequence of compensation activities, or a dynamically generated sequence, is executed without much concern for the resulting cost, handling an exception may prove more costly than not handling it. However it may be necessary to execute some compensation activities in order to evaluate the cost. Yet if an exception is rejected then the effects of executed compensation activities must be eliminated (i.e. be compensated for themselves). This ideally involves recursively (1) identifying those that activities have been affected for a given exception and determining the total impact of that exception, and (2) evaluating whether other properties for the exception will reduce the total impact.

General requirements identified by 1, 2 and 3 above can be used to make absolute judgements about the completeness of actual capabilities included into the specification of a sales order processing workflow system. Also relative completeness judgements can be made in order to help select the best available workflow approach, or indeed to devise an improved approach, for use in the sales order processing domain.

Consider for example the control flow specification from Figure 50.A and the control flow instance from Figure 50.B. The control flow instance indicates that activities A, B, C and G have been successfully executed and activity D and H can be started at any point in time. Note that activity G has been performed conditionally as the predicate $Q > 110$ is valid since $110 > 100$. Assume that now an exception occurs. In this case:

1. Activity G needs to be cancelled, as now Q equals 90 and therefore a business rule is no longer valid (i.e. precedence link PL8 is not enabled anymore).

2. Activity C needs to be updated since its input has changed, if we assume that activity A generates data for activity C.

3. Activity D and H should be blocked since (A) the input of activity D is uncertain and activity D can be started at any time by the user, and (B) the need to perform activity H is uncertain as precedence link PL9 is not enabled under conditions where an exception has occurred.

Activity B has not been affected by the exception, i.e. it does not need to be updated nor to be cancelled. Thus precedence link 8, activity G, precedence link 9 and activity H comprise an invalid path of execution under conditions where an exception has occurred. Precedence link 1, activity A, precedence link 2, activity B, precedence link 3 and 4, and activity C and D still form valid paths of execution.

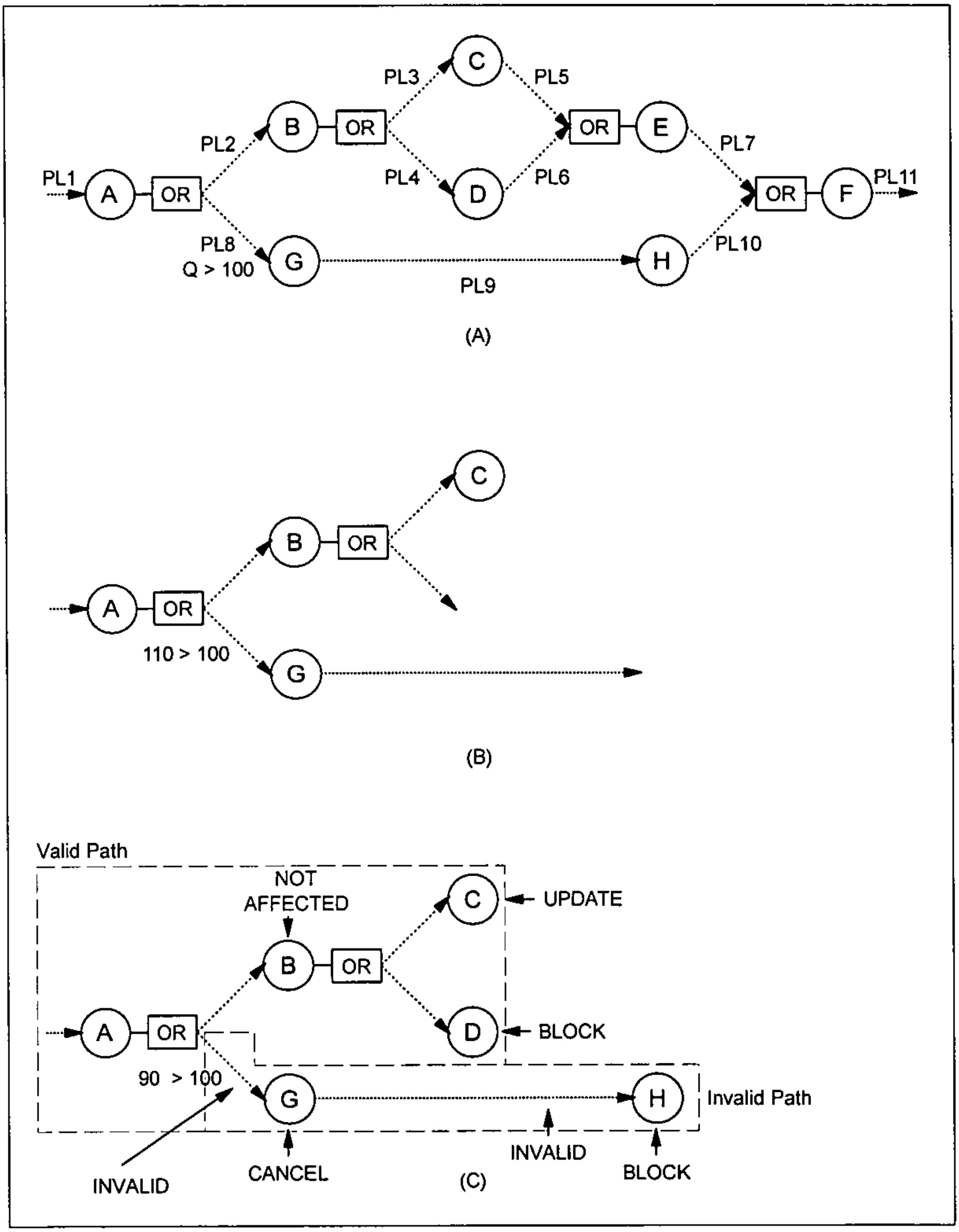


Figure 50: Example effects of an exception on the state of a workflow.

3.2 Effort and flexibility competences

3.2.1 Effort competence

Focus of study in this thesis is on the engineering effort and time needed to define appropriate properties of exception handling specifications for a given workflow and its exceptions using a modelling philosophy and modelling constructs provided by a workflow approach. In general the development of exception handling specifications will always require the identification of types and instances of exceptions that may occur within a given workflow, and an analysis of their effect on previously performed activities whatever the progress of a workflow (as discussed in section 2). Yet workflow approaches adopt different philosophies with respect to harnessing and exploiting knowledge when handling exceptions. Indeed, here particular interest was centred on the time required to define exception handling specifications according to the philosophy of (A) specifying explicit, static description of sequences of compensation activities for all possible workflow states or (2) dynamically generating sequences of compensation activities from encoded knowledge about a system (i.e. via the use of notions such as 'spheres' in Partial Rollbacks) that exploit control flow dependencies.

Here it was taken as a 'given' that the scale of effort will be influenced directly by properties of (a) the modelling philosophy and modelling capabilities provided by a workflow approach and system and (b) the complexity of the real system in which workflows are to be managed and controlled. Further modelling properties referred to under (a) will impact on the required scope (breadth) and detail (depth) of approaches and on the range and types of application area in which the approaches can be successfully deployed. While the complexity of the real system will be influenced by many factors such as the number of workflow activities involved, the complexity of these elemental activities, the complexity and predictability of cause and effect relationships and flows coupling activities, and so forth.

The criterion effort is an important measure in situations where workflow approaches offer similar functional capabilities, yet the time to develop exception handling specifications for the same workflow is very different when deploying alternative workflow approaches. It is worth noting that workflow approaches cannot be compared on the criterion of effort alone since approaches may serve different activity synchronisation problems. Doing so may amount to 'comparing apples and oranges'. However for approaches that address similar activity synchronisation problems the effort required to implement a model can be an important issue, as will be shown later in this study.

3.2.2 Flexibility competence

Particular focus has been on facilitating response to uncertain workflow requirements change that takes the form of unpredictable occurrences over time of common exception types and conditions.

Within the context of this study, the notion of flexibility, and means of quantifying the inherent flexibility of workflow approaches, has been linked to issues related to specific effort required to update standard workflow and exception handling specifications when predictable and unpredictable forms of change occur. Typical predictable changes that may occur to a standard flow of control[29] are (1) the introduction of new activities, (2) the deletion of existing activities, and (3) changes to the sequence in which activities are executed. As a result the number of new states and attributes of states a standard workflow may enter will be determined and implemented by making changes to the standard workflow specification. Exception handling specifications are based on exceptions that may occur and the states a workflow may enter. As a consequence a large number of exception handling scenarios may need to be determined, modified and enacted. Evidently therefore there can be a very rapid growth in workflow complexity, following growth in the number of activities, interconnectivity of activities, the number and variety of exception occurrences, and so forth.

Flexibility by its nature might be coupled to the criterion effort. Hence it is emphasised that the specific capability of a workflow approach to limit or constrain the impact of a change (e.g. to a standard flow of work or to parts of exception handling specification or model instance) is likely to yield improved flexibility. On the contrary, if a change impacts extensively on a workflow specification then an approach may be considered to be inflexible.

3.3 Correctness competence

Within the context of this study, a focus on correctness issues has centred on investigating the following assumptions:

1. That workflow designers often make mistakes when capturing and developing exception handling specifications and that inherent capabilities of a workflow approach will influence the extent to which these mistakes will limit the utility of the approach.
2. The effort required by a workflow designer to produce a sufficiently correct workflow specification will be determined by the way exception handling specifications are formulated.

Whilst investigating assumptions 1 and 2 it was found to be important to recognise that the provision of generic modelling constructs (as opposed to application specific ones) will bring

---

[29] It must be noted that a change to a type of business rule is different from a change to the state of a business rule instance, although both changes constitute exceptions in their own right. Whereas the former change is concerned with the order in which activities of a workflow are executed in general (and therefore with the order of execution of all instances that are based on this type), the latter change is concerned with the order of execution for a specific workflow instance within the context of rules defined by the specific workflow type.

advantages and disadvantages. A suitably defined set of generic modelling constructs might, for example, be capable of facilitating the specification of exception handling routines in various application domains and thereby improve the flexibility of the workflow approach in terms of its breadth of applicability. On the other hand in any specific application, use of generic constructs may prove more difficult than domain specific ones, as their use may be open to interpretation / misinterpretation by the workflow designer. Consequently it may be relatively easy to make mistakes when formulating and changing specifications and it may be a difficult and tedious job to check the consistency of those specifications. From a theoretical point of view, if a workflow designer has ample expertise and is given sufficient modelling time, then each instance of a model definition will be correct. However it is well known that humans are prone to making mistakes when their task is lengthy and monotonous[30] [31].

Although correctness criteria are considered to be of importance, this research study majored on the development and use of 'completeness' and 'flexibility' competance measures. None the less, the study did consider workflow system correctness issues particularly during the architectural design and development of the new workflow approach. Based on this work, later thesis sections recommend that workflow capabilities suitable for SOP application domains should incorporate means of facilitating consistency checking.

3.4 Discussion

In this chapter a list of requirements was derived from the characteristics of sales order processing. This study has defined general end-user requirements, namely by analysing and abstracting requirements stated in other publications. For example in many publications the need to redo activities has been stressed. However by considering properties of the target domain this study has identified common properties of activities that must be redone and how that might be systemised (e.g. with reference to the existence of data and control flow dependencies). A similar argument was developed for the requirement to temporarily block the execution of activities. Furthermore a new end-user perspective on requirements was formulated that states that a started exception handling procedure may need to be (partly) cancelled.

---

[30] Experience gained in this study showed that correctness aspects of workflow approaches alone cannot be compared with each other unless the use of the approaches is compared for a similar activity synchronisation problem.

[31] It is necessary to distinguish between a modelling method and a method used to check the consistency of a model. Many workflow systems come with an explicit modelling method or an implicit one that is supported by one means of its techniques. However a workflow system is less likely

With respect to the needs of workflow system developers, in the last decade significant research effort has been directed towards the analysis and development of workflow approaches from a system developers viewpoint. Indeed three main strands of new research were identified, namely 'workflow activity failures', 'workflow correctness' and 'workflow flexibility', that were connected by the common theme of exception handling. Even though exception handling has been a focus of concern for many previous research studies, this study has developed a new perspective on this broad topic. This new view has been developed by investigating the hypotheses that (1) within a workflow specification it is possible to distinguish a single standard workflow specification and a set of exception handling specifications and (2) changes to any standard workflow specification can result in change to various instances of exception handling specifications. In the sales order processing domain three common classes of change to standard workflows were identified and characterised, namely (1) insertion of workflow activities (into workflow specifications), (2) deletion of workflow activities and (3) change in the order of activity execution.

Subsequent chapters of this thesis develop and test aspects of these hypotheses, where testing is focussed mainly by considering two criteria namely:

1. 'Correctness'. To date correctness of workflow specifications has been largely researched within the context of traditional workflow approaches that are based on precedence link, junction and activity modelling constructs (see section 3.3 of chapter 2). In these studies two kinds of mistake made by designers and modellers were recognised leading respectively to 'deadlocks' and 'lack of synchronisation'. However other classes of workflow approaches provide modelling constructs that can be combined in various ways by workflow designers (e.g. 'event-condition-action workflow approaches' and 'transactional workflow approaches'). Consequently the impact of different modelling mistakes (made by workflow designers) is dependent on the type of workflow approach used. This study contributed new understanding in this area. It lifts the issue of correctness to a more abstract level and it argues that first of all appropriate workflow approaches must be developed that meet end-user and workflow system developer requirements, before the attention should be focused on detecting and analysing designer induced modelling mistakes.
2. 'Effort & Flexibility'. Flexibility is a term that has had various meanings in the previous literature on workflow systems. It implied organisational changes (see section 3.6 of chapter 2 'change capable workflow systems') and exceptions within an executing workflow (see section 3.1 of chapter 2 'workflow activity failures'). Each of these flexibility types is needed

---

to be supported by a method for checking the consistency of models specified and enacted by that workflow system.

for a purpose and thus should be provided as a capability of a workflow system as required. In this study focus has been on the criterion 'effort' which is related to 'flexibility'. Whereas flexibility can be considered to relate to the ability of reconfigure the workflow specifications during the system maintenance phase here it is argued that effort is concerned mainly with workflow configuration during the system development phase.

6. Detailed analysis of the capabilities of existing approaches to workflow

1.0 Introduction

This chapter considers to what extent the sales order processing domain requirements described in chapter 5 can be satisfied by functional capabilities incorporated into the current generation of workflow systems reviewed in chapter 4. The purpose of this capability analysis is to identify strengths and weaknesses of the current provision of workflow concepts and techniques.

2.0 Traditional workflow approaches

Traditional workflow approaches can detect logical failures of activities when they complete and trigger specific exception handling procedures. Such procedures contain (A) predefined compensation activities that undo the results of previously performed activities and (B) other activities that achieve the objective of the workflow but in a different way. An alternative for part (B) is to use a loop construct to let a thread of control flow back to an upstream activity so the standard procedure can be repeated. By taking into account knowledge about the expected progress of a workflow thread a designer can, at design time, explicitly define an exception handling scenario for those activities at which execution can fail logically. Consider for example the SOP workflow specification from Figure 51. Two common types of activity in the SOP domain that can fail logically are the activities 'Allocate Stock' and 'Check Creditworthiness of Customer'. Here however only the handling of a logical failure of an 'Allocate Stock' activity is considered. A logical failure of the 'Allocate Stock' activity can be resolved in two ways, viz.: (1) by making a change to the MPS or (2) by making a change to the sales order[32]. Thus by embedding a new activity, 'Modify Sales Order', into the original workflow specification such an exception can be handled in the manner illustrated by Figure 51.

---

[32] As a change in the MPS requires the workflow to cross a process boundary (a situation that will not be considered in this thesis) here only considered is a change to the sales order so that the issue can be resolved.

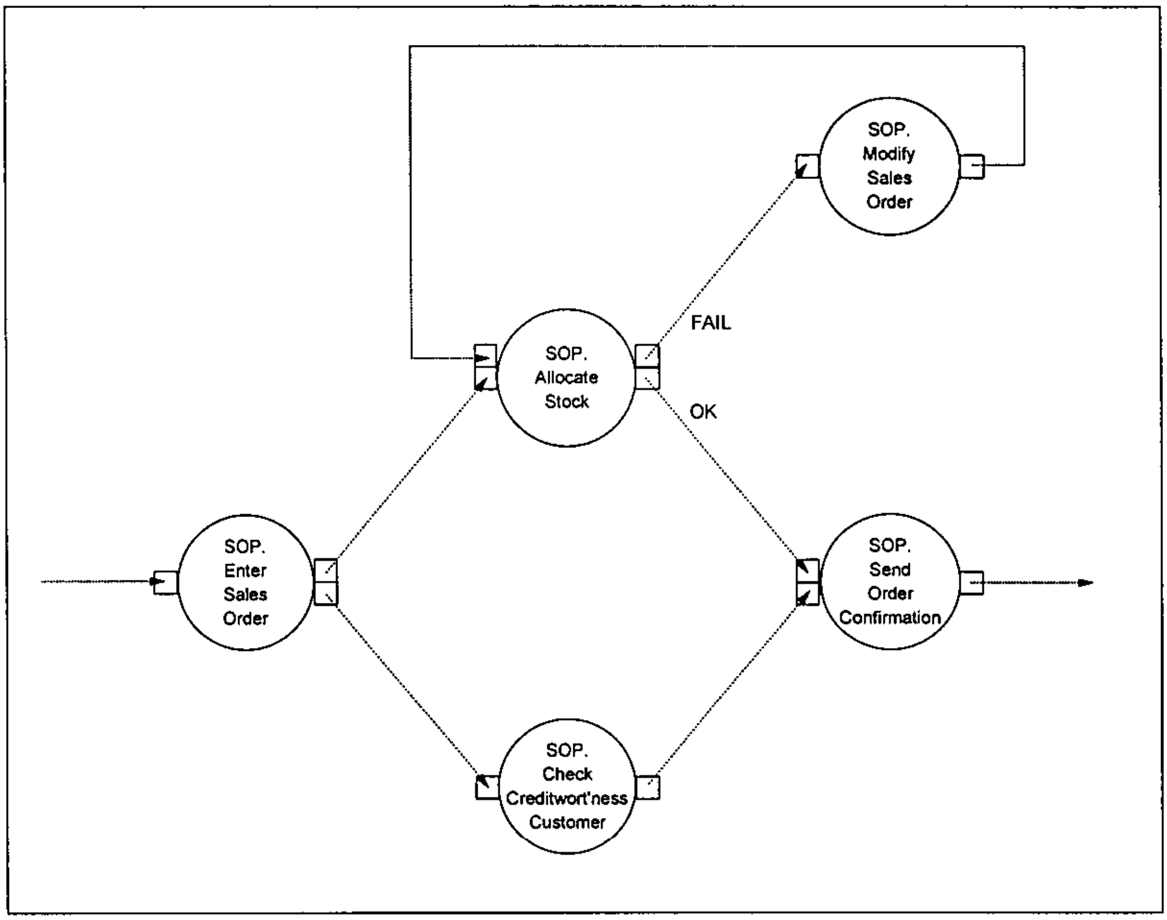


Figure 51: Example workflow specification which includes an exception handling procedure to cope with a logical failure of the 'Allocate Stock' activity.

2.1 Completeness

Traditional workflow approaches can detect logical activity failure exceptions. However, they do not provide means of modelling events that correspond to a 'change to the input of a completed activity' nor do they provide direct means of dealing with 'cancellation events' or 'business rule exceptions'. Traditional approaches to workflow only recognise event types for which an associated activity completion is assumed either to be successful or result in a 'logical failure'. The latter case is treated as an exception. Indeed in traditional workflow approaches control is passed to those activities that are being executed.

To some extent, at the time of design a workflow designer can determine and reason about those activities that may need to be updated, cancelled or required to be temporarily blocked in order to define and embed exception handling procedures into the workflow specification. Typically the workflow designer can reason about properties of exceptions and the workflow state and conditions under which activities will have been performed. However the workflow designer may be hampered whilst developing an appropriate compensation response if a workflow has multiple parallel workflow threads. Generally if an activity in any single path can fail logically then the workflow designer will be able to plan and define the state for that thread. However they will not be able to develop such a definition with respect to other related threads of execution. The other threads of execution may have reached different stages of progress (i.e. could just have been started or be close to completion). It follows that in general a workflow designer needs to pre-define complex exception handling scenarios for all possible workflow states and this may well prove impracticable.

Consider for example a workflow that has two concurrent paths of execution, as follows: path 1 consists of activities B and C and path 2 consists of activities D and E. Such a workflow example is illustrated by Figure 52. Assume that the execution of activity C can fail. The designer can be sure that activity A and B must have been performed if activity C fails and that activity F will not yet have been performed, whereas the designer will be uncertain about the progress of thread 2 which might have reached different stages at instances when activity C can fail. Activity D and E may yet have to be performed, or activity D may have been performed but not activity E, or activity D and E may have been performed.

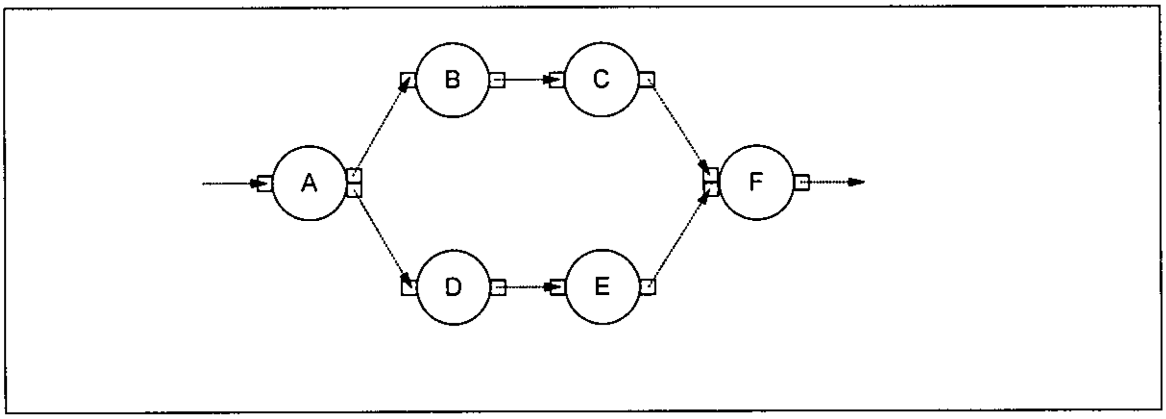


Figure 52: Workflow specification with two paths of execution.

Another problem that becomes clear upon detailed analysis is that by passing control from the failed activity 'Allocate Stock' onto the activity 'Modify Sales Order' (so that the exception can be handled) the latter activity does not have sufficient knowledge about how the sales order should be modified so that the 'Allocate Stock' activity can complete successfully next time. The workflow approach only triggers a signal to perform an activity yet does not indicate how. Of course if both activities are performed by the same person then that person has the knowledge needed to recover from the exception. However if the two activities are performed by different employees at different locations they will need to communicate with each other about particulars of conditions related to the exception.

Traditional workflow approaches cannot reverse the effects of exception handling procedures that are being executed when other exceptions occur, unless their occurrence has been anticipated by workflow designers. In order to reverse such effects, additional compensation activities will need to be specified that compensate for the effects of already executed compensation activities associated with a current exception handling procedure. Yet it is very difficult to actually implement such actions as so many exception variations may arise while executing an exception handling procedure.

2.2 Effort and flexibility

We may deduce that the effort required to establish an exception handling specification will increase with the number of activities that can logically fail and the total number of activities in a workflow specification. For each activity that can fail it is necessary to identify those workflow activities that may have been impacted on before an exception handling specification can be established. Consider for example two (linear) workflow specification such as: (1) 'Enter Sales Order', 'Pick Goods', and 'Invoice Customer', and (2) 'Enter Sales Order', 'Allocate Stock', 'Check Creditworthiness of Customer', 'Send Order Confirmation', 'Pick Goods' and 'Invoice Customer', where the activities 'Allocate Stock' and 'Check Creditworthiness of Customer' may be performed in parallel. In case (1) only the activities 'Pick Goods' and 'Invoice Customer' can fail and the handling of these exceptions is simple. For example if activity 'Pick Goods' fails and not all items can be picked, either the customer accepts those items picked or all items are returned to stock. In case (2) many activities can fail and many different ways can be chosen to resolve the exception.

The flexibility of traditional workflow specifications can be shown in general to be low. If an activity is to be removed from or introduced into the path of any activity that can fail or the order in which these activities are executed, generally this will impact on the possible states of progress of a workflow and the exceptions generated. In turn the structure of existing exception handling specifications will have to change. Logically one might assume that the effort involved in changing a workflow specification will be inversely proportional to both (1) the number of activities contained in a workflow specification and (2) the number of activities that can fail logically. The greater the number of activities that can fail logically (or the greater the number of exceptions individual activities can generate) and an increased number of possible states a workflow will have, generally the larger workflow exception handling specifications will become. As a consequence there may be significant time and cost incurred when making a specification change. However the previous literature does not provide any formal proof to back up the practical observations made, nor does it give any real indication of where (and where not) these approaches can successfully be deployed. Indeed with traditional workflow specifications there is no embedded notion about the re-use of exception handling scenarios so that every installation at a customer site and indeed different instances of a given installation at a customer site will require a new analysis and a new implementation of an exception handling specification.

To illustrate this point, consider the example workflow where activities such as 'Enter Sales Order', 'Send Order Confirmation' and 'Allocate Stock' are performed sequentially. Execution of the activity 'Allocate Stock' can fail and a suitable exception handling scenario may be 'Modify Sales Order', 'Re-allocate Stock', 'Send Order Re-confirmation' (if Re-allocation of Stock is successful). Now consider a change to the standard order of execution where the activity 'Allocate Stock' is performed before the activity 'Send Order Confirmation'. This will invalidate the existing exception handling specification. A similar impact will result when deleting the activity 'Send Order Confirmation' from the workflow specification or when inserting the activity 'Check Customer Creditworthiness' into the workflow specification.

2.3 Correctness

As a result many practical exception handling specifications are known to become overly complex. Moreover traditional workflow approaches do not have an underpinning formal theory that can be used in a systematic way to analyse, identify, and predict possible exception handling scenarios for different workflow states. They have no associated laws or rules defined that can be used to determine those threads of execution that may be (partly) invalid when an exception occurs. When using traditional workflow approaches, at best a workflow designer can only take into account the state of activities in the activity path that has failed by referencing the workflow specification and specifying appropriate sequences of compensating activity to handle the exception. However information available during design may be incomplete and as a consequence: (1) necessary compensation activities may be left out, (2) wrong compensation activities may be specified and (3) redundant compensation activities may be included.

Consequently with increase in complexity of the workflow specification is likely to come a sharper increase in the likelihood that the workflow designer can make mistakes. The cost of an incorrect workflow specification is that sometimes unsatisfactory conditions (e.g. that a customer is not creditworthy, or that insufficient components are available) may go unnoticed for a long time resulting in high cost as a consequence of needing later to correct the situation. This observation will be conditioned by the experience, intuition, intelligence of a workflow designer but in general traditional workflow approaches will not be robust in correctness terms when used in many practical application areas such as our idealised sales order processing domain.

3.0 Event-Condition-Action workflow approaches

The power of Event-Condition-Action workflow approaches lies in the fact that they can capture many kinds of event, define and detect many kinds of workflow states, and tie them to various sets of necessary actions. With respect to specifying the order of activity execution ECA approaches do not really distinguish between standard flows of control and exception flows. Rather they are dealt with by virtually the same means, namely by applying event-condition-action rules.

Consider for example the standard flow of control in the simple workflow illustrated by Figure 53. If case A occurs (i.e. activity A has been completed) and variable A.Q is larger than 100 then case B is to be performed. However if A.Q is smaller or equal to 100 case C rather then case B is to be performed. When either case B or C has been completed case D is to be executed.

(1) CaseEnd(A); A.Q > 100; start(B)

(2) CaseEnd(B); A.Q <= 100; start(C)

(3) CaseEnd(B); ; start(D)

(4) CaseEnd(C); ; start(D)

Figure 53: Standard flow of control as defined by ECA-rules.

Using ECA approaches it is possible for a given workflow to define all types and instances of exceptions, the various workflow states that can exist as exceptions occur, and ways of handling exceptions as and when they occur. Consider the case where an exception occurs because of a data modification to variable A.Q. At the point in time when this exception occurs the workflow may have reached different states (as defined from a process perspective), namely: case A may have been performed; case A and B may have been performed (so that implicitly A.Q > 100); case A, B and D may have been performed; case A and C may have been performed (so that implicitly A.Q <= 100); or case A, C and D may have been performed. The exception handling specification required to deal with this exception type at different stages of workflow completion are therefore defined in Figure 54.

(1) modify (A.Q); performed(B), not_performed(D), Q > 100; update(B), suspend(D)

(2) modify (A.Q); performed(B) , not_performed(D), Q <= 100; cancel(B), suspend(D)

(3) modify (A.Q); performed(C) , not_performed(D), Q > 100; update(C), suspend(D)

(4) modify (A.Q); performed(C) , not_performed(D), Q <= 100; cancel(C), suspend(D)

Figure 54: Exception handling specifications as defined by ECA-rules.

3.1 Completeness

From this simple example, one can deduce that:

1. ECA workflow approaches recognise a wide range of exception types. Included within such a list of exception types that can be modelled are generalised exceptions that can be used to model (A) a change to the input/output of a completed activity, (B) a logical activity failure and (C) a business rule change.

   Comments about generating ECA approaches of these general exception types are made in the following.

   A. Change to an input/output. Although this class of exception may not be explicitly supported by specific event-condition-action workflow approaches, it is observed that WIDE can encode exceptions in terms of modifications to object attributes. This is similar to encoding exceptions arising because of a change to the input/output of a completed activity. Therefore potentially techniques used to encode properties of exceptions arising from modification to object attributes may be appropriate for use when modelling exceptions arising because of input/output change.

   B. Logical activity failure. Modelling of this class of exception is supported by all event-condition-action workflow approaches described in the literature. The WIDE approach detects the completion of an activity by means of a caseEnd event and determines the success or failure of activity execution by use of an 'object.variable = constant' predicate in a condition. The Rapide approach determines the success or failure of activities in a similar way.

   C. Business rule change. Event-condition-action workflow approaches do not recognise a state change in a business rule as a specific class of exception. Although the WIDE approach can detect changes to the state of object variables this technique can be used to model business rules in part[33]. But the modelling of some external events as supported by WIDE techniques can be used to represent special business rules such as the need to start or cancel the execution of a workflow or workflow thread.

2. ECA workflow approaches do not provide specialised modelling constructs or enactment mechanisms to automate the determination and handling of activities that need to be updated, cancelled or blocked in situations where an exception occurs. Yet workflow designers can use general ECA rules to specify needed compensation activities for a given instance of an

---

[33] In situations where business rules are composed of multiple variables, a change in state of one of them does not necessarily have to trigger a business rule state change.

exception type and workflow state of progress. Consequently it is left to the workflow designer to determine (for a given kind/instance of exception and workflow progress) those activities that will be affected and what and how compensation actions should be achieved. Therefore when using ECA approaches the workflow designer must reason about characteristics of activities, the characteristics of the exception type and instance, the state of progress of a workflow (from both data and process-centric viewpoints), and about the flow of control amongst dependant activities. When so doing experimental work in both Sales Order Processing domains, the author has observed that the use of ECA rules can have three important limitations, namely:

A. Because the properties of an update and cancel compensation activity cannot explicitly be determined (i.e. as actual changes to inputs or outputs are not numerically evaluated) then the user is simply instructed to update or cancel an activity. The user is not helped as to how this should be achieved.

B. The execution of compensation activities cannot be explicitly sequenced. Rather it is assumed that all compensation activities specified are executed concurrently, while preserving certain producer/consumer activity dependencies between up- and down-stream activities. Consider for example use of an ECA-rule where activities B and D both need to be updated. Yet (1) it may be that activity D uses data generated by B and (2) activity D may complete before activity B has generated its output data.

C. Activities that have been suspended upon the occurrence of an exception cannot be released following the resolution of any uncertainty.

Because of limitations of ECA modelling techniques, in practice typically a workflow designer will only specify exception handling scenarios for the most frequently occurring exceptions and the most common states of workflow progress; less common situations are therefore left to be addressed outside the scope of the workflow system. It follows that for practical reasons workflow specifications developed in conformance with event-condition-action workflow approaches are often far from complete.

3. ECA workflow approaches simply execute actions specified by the action part of an ECA rule. They do not provide capabilities to reverse the effects of executing such actions. However to some extent the workflow designer can reason about the impact of particular exceptions and the required compensation activities, and identify those exception handling scenarios that can be executed without concern. Moreover it may be possible to specify a set of alternative compensation activities that may be used, such as in cases where the handling of an exception becomes too costly.

3.2 Effort and flexibility

It can be observed that the effort required to define ECA exception handling specifications will be linearly related to the number of (A) states that a workflow may reach and (B) exceptions that may occur. In a way similar to traditional workflow approaches it is necessary to define for every exceptional situation (as defined by a combination of an exception that may occur and a workflow state) a unique exception handling specification. As a consequence problems may arise from 'exploding' exception handling specifications and resultant exception handling specifications may well become more complex than the standard workflow specification itself. Generally it has been found that establishing event-condition-action workflow specifications requires a large and time consuming effort.

ECA-exception handling specifications are generally inflexible. If the standard order of execution is modified (i.e. activities are deleted, inserted or their order is changed), it is likely that the states that a workflow may reach will be different. Yet all exception handling specifications are based on these workflow states. Consider for example the effect of removing an activity from the standard workflow specification. This will impact directly on the possible states that the workflow can reach as fewer activities will need to be performed for a workflow to complete its execution. Yet activities from exception handling specifications that compensate the results of this activity, will also need to be removed. It follows that significant effort will be required to modify ECA workflow specifications even should only a single activity be deleted. Essentially this situation arises as a consequence of needing to produce a unique exception handling specification for every exceptional situation. Although there will be similarities between the exception handling scenarios there will also be differences. Indeed exception handling scenarios cannot be re-used in other specifications.

3.3 Correctness

Event-condition-action workflow approaches are not underpinned by a formal theory. If a change to the state of a previously performed activity occurs (i.e. either a change to an incoming flow of control or an input change occurs) there are no general rules that can be applied that help determine which downstream activities will be affected and/or determine what effect the exception will have on the activity output (i.e. whether the output should be updated or whether the output is no longer required). The same argument applies to an exception arising from a logical activity failure and its effect on upstream activities. The extent to which a complete workflow specification (including predefined standard business logic and exception handling specifications) is correct will depend largely on the knowledge, experience and intuition of the workflow designer and on the complexity of the standard business logic required (i.e. on the number of activities, number of paths of execution and number of exceptions that may occur). In order to specify appropriate compensation actions for all possible exceptions a workflow designer has to evaluate all possible paths of execution. Generally this will mean that some necessary compensation activities will be left out and inappropriate ones may need to be excluded.

4.0 Transactional workflow approaches

Transactional workflow approaches do not explicitly specify detailed sequences of compensation activities such as those developed as part of traditional and event-condition-action workflow approaches. Rather they attribute a compensation activity to each workflow activity and define particular rules whereby a "standard" workflow specification should be interpreted under conditions that can exist when exceptions occur. At execution time transactional workflow approaches dynamically generate the required sequences of compensation activities needed to handle an exception. This dynamic generation of sequences of activities is derived from (1) sphere definitions and (2) control flow dependencies linking activities or activity status information. Figure 55 graphically represents example standard activities found in the idealised SOP domain and attributes compensation activities to them. Furthermore, the sphere that encompasses activities 'Enter Sales Order', 'Allocate Stock', and 'Check Creditworthiness Customer' defines that if any of these activities may fail, they all will need to be compensated.

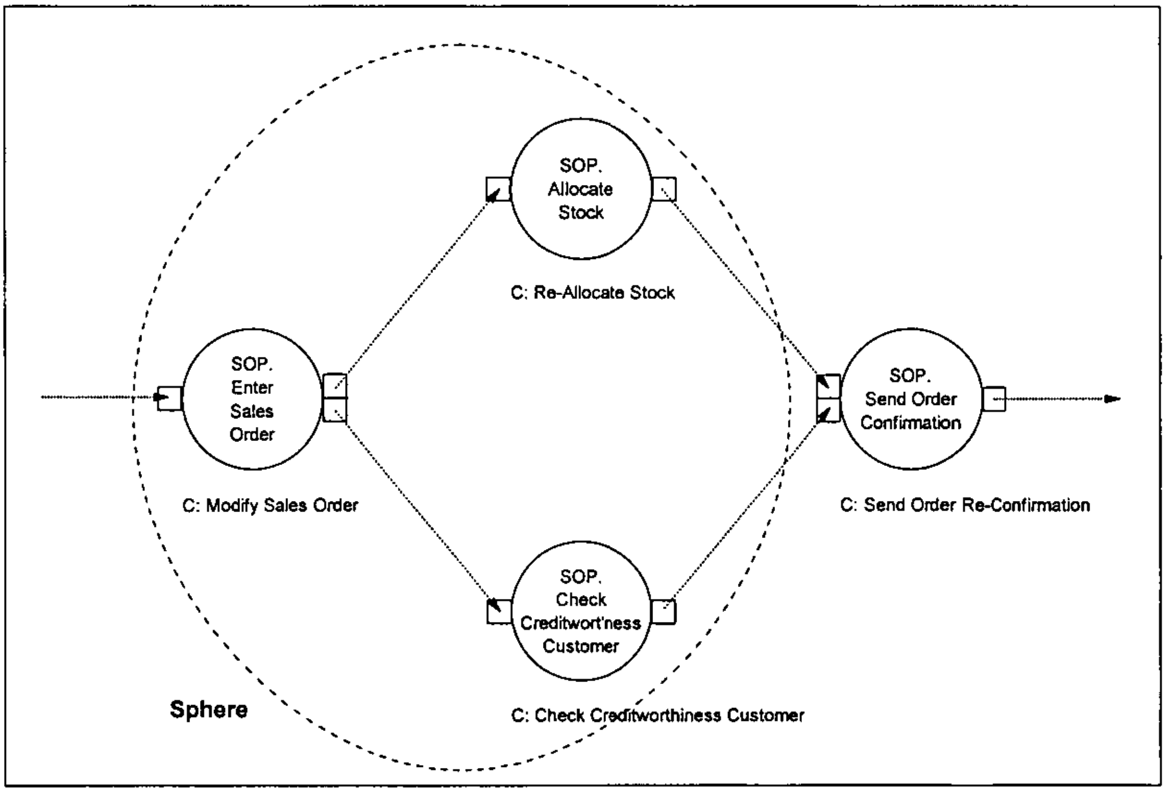


Figure 55: Transactional workflow specification for sales order processing.

Assume that at the workflow progress shown in Figure 56 insufficient stock can be allocated to the order. This will require the activities that belong to the sphere to be compensated if they have been performed. Thus compensation activity 'Modify Sales Order' will be triggered. Next the compensation activities related to the 'Allocate Stock' and 'Check Creditworthiness Customer' activities are invoked, i.e. the activity 'Re-Allocate Stock' and 'Check Creditworthiness of Customer'.

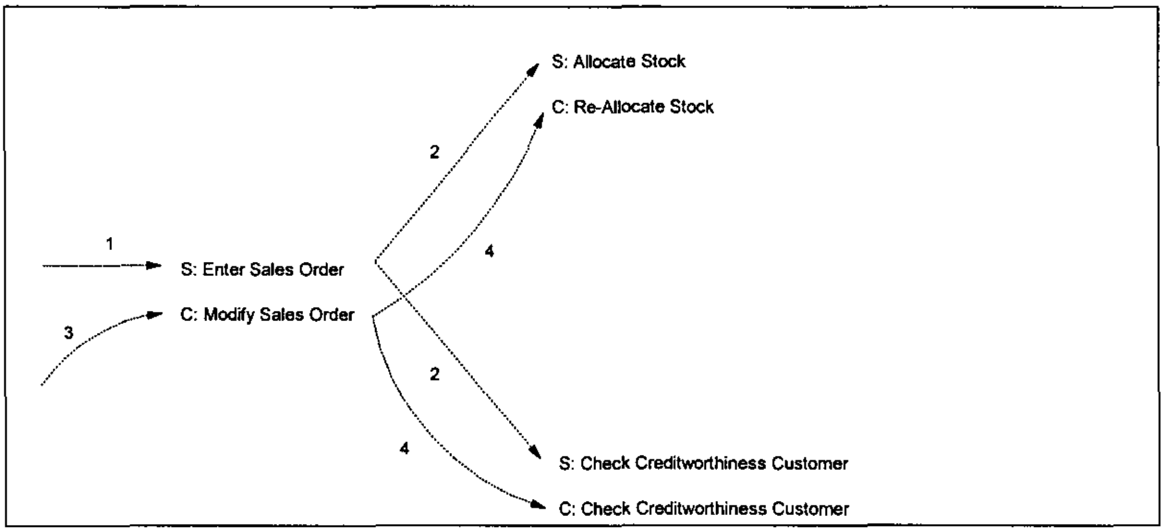


Figure 56: Scenario of issued standard and compensation activities.

4.1 Completeness

Transactional workflow approaches have the following capabilities and limitations:

1. The exception types 'input change' and 'business rule change' cannot be detected and handled by transactional workflow approaches. The only type of event that can be detected is the completion of an activity, whether this is successful or whether it fails.
2. In case such exceptions occur it is possible to compensate activities in a sphere. Yet not always will executed standard activities require to be redone. E.g. if a customer decides to order fewer products and the activity 'check creditworthiness customer' has already been performed, it does not have to be redone. Further it is only possible to attach either an update or a cancel compensation activity to any standard activity, limiting more complex application scenarios where under certain conditions activities results may need to be updated or need to be cancelled. However in contrast to traditional workflow approaches, transactional approaches are capable of updating activity results for workflows with multiple paths. Yet these approaches are not capable of detecting invalid paths of execution. Finally when handling exceptions transactional workflow approaches do not provide capabilities to block activity execution. When compensation activities are executed, the standard workflow may continue.
3. The effects of compensation activities set into motion to handle an exception cannot be reversed. At first sight it seems that the appropriate exception handling scenarios will be generated for every state of a sales order workflow. However if the execution of some of the

compensation activities fails (such as for compensation activities 'Re-Allocate Stock' and 'Check Creditworthiness Customer'), a workflow can still reach an inconsistent state. Therefore to correct failures arising during compensation activities additional compensation activities will need to be performed outside the scope of the formalised workflow system.

4.2 Effort and flexibility

In comparison with traditional and event-condition-action workflow approaches the effort involved in establishing a transactional workflow specification is relatively low. This is because (A) for every standard activity in a workflow specification an appropriate compensation action only needs to be specified and attributed to that activity, and (B) spheres need to be specified for activities which results are logically related. There is no need to specify explicit flows of exception handling specifications with the transactional workflow paradigm. Thus in the idealised sales order processing domain there will be a total of 4 compensation activities that need to be specified when adding compensation activities logic to a transactional workflow specification.

In principle transactional workflow approaches are inherently flexible. A change to the workflow specification (such as the order in which activities are executed and/or the insertion and deletion of activities) should result in very little additional work. Firstly there is no need to change compensation activities associated with other standard workflow activities. Secondly, however, spheres may need to be revised but this is little work when compared to making changes to explicit sequences of compensation activities. It be argued that it may be sufficient to illustrate the high potential for flexibility by way of an example. Consider for instance the deletion of an activity such as 'Check Creditworthiness Customer' from the workflow specification. Although some precedence relationships must be redefined there is no need to change compensation activities associated with other standard workflow activities. Indeed transactional workflow approaches replace the use of explicit pre-established exception handling specifications by the use of general workflow rules. Theoretically therefore, by adopting a transactional paradigm, workflow systems can be reconfigured readily in response to business requirements change.

This flexibility, however, is obtained through imposing a particular synchronisation structure onto activities. Part of this synchronisation structure is implicitly determined by general properties of the modelling enactment techniques rather than by specific organisational needs. The use of such an approach must incur a risk that it may not prove possible to capture certain semantics of any specific synchronisation scenario. Whereas particular types of activity synchronisation may be supported in a very flexible way, it may constrain or even prevent other types of synchronisation. Such constraints are evident when the transactional approaches adopted by Sagas and ConTracts are considered. They require complete sets of previously performed activities to be compensated for upon the occurrence of a logical activity failure. The subsequent development of the Partial Rollbacks approach recognised that the results of some previously performed activities might remain valid. This points up a key precept when developing next generation workflow paradigms, namely that flexibility should be achieved whilst making sure that selected workflow rules closely fit the synchronisation requirements of the application domains in which the new paradigm will be used.

4.3 Correctness

Although it has been a principal aim of transactional workflow approaches to develop and apply a formal theory that can underpin the execution of activities, in reality few of these approaches are provided with any explicit statement of their underpinning theory. Usually in some way the rules are embedded into the workflow approach itself. The rules are based on the concept that activities that have actually been executed should be partly or completely compensated for in the originally specified order of execution. Although this represents a significant step forward in defining a theory that can underpin an effective and practical exception handling approach it will still have a number of limitations when it is applied in the domain of SOP-applications.

5.0 Comparison drawn between the workflow approaches reviewed

This section interprets comparative observations about existing workflow approaches individually described in the foregoing. Focus of attention is on tabulating comparative features of existing approaches so as to gauge relative measures of their inherent capabilities (as illustrated in Table 6).

5.1 Completeness

It can be observed that the *logical failure class of exception* is supported by all three types of workflow approach previously considered. However neither the *change in input/output* nor the *change in state of a business rule* exception class is explicitly modelled by any of these three types of workflow approach. On the other hand, event-condition-action workflow approaches do generally provide modelling capabilities that can be used indirectly to encode properties of these other exception classes by modelling so called object attribute values.

It can also be observed that 'transactional' approaches posses a capability to dynamically generate compensation activities. These approaches encode some sort of exception handling knowledge. However this knowledge is only sufficiently rich to generate update compensation activities for a complete set of previously performed activities[34]. In practice some of these previously performed activities may not have been affected and therefore do not require costly and time consuming

---

[34] Although the Partial Rollbacks approach can either generate update or cancel activities for affected activities.

compensation actions. Further these approaches cannot dynamically determine those activities which execution is to be blocked.

On the other hand workflow approaches that execute predefined sets of compensation activities (such as traditional and event-condition-action workflow approaches) are capable of executing update and cancel compensation activities by referencing a given exception and workflow state, as well as block the execution of activities for which the validity of input values may be uncertain. That said, none of the existing approaches can automatically determine all necessary parameters of compensation activities.

Also, none of the approaches reviewed provides capabilities to reverse the effect of executed compensation activities. Rather the effects of compensation activities are immediately made permanent. Traditional and event-condition-actions workflow approaches can specify additional compensation activities in predefined exception handling scenarios with capability to undo the effects of previously performed compensation activities, but such a technique necessitates ‘correction of corrections’.

Criterion			Traditional workflow approaches	Event-Condition-Action workflow approaches	Transactional workflow approaches
Completeness	Exception Types	Logical failure	Yes	Yes	Yes
		Change to input/output	No	Yes	No
		Business rule change	No	Partly	No
	Execute appropriate compensation	Dynamic generation of update and cancel compensation activities and identification of activities to be blocked	N.A.	N.A.	Partly, either update or cancel
		Static specification of update and cancel activities and activities which execution is to be blocked.	Yes	Yes	N.A.
	Reverse effects of executed compensation activities		Partly	Partly	No
Effort & Flexibility		Effort	High	High	Medium
		Flexibility	Low	Low	Medium
Correctness			Low	Low	Medium

Table 6: Tabulated comparison of the capabilities of current workflow approaches, as reported in the literature and reviewed in this chapter with reference to semi-generic sales order processing needs.

5.2 Effort and flexibility

The effort involved in defining exception handling specifications for event-condition-action workflow approaches is high as this class of approach requires explicit sequences of compensation activities to be defined for every possible exception and workflow state. Moreover it is not possible to re-use parts of these predefined sets of compensation activities in different exception handling specifications. Comparatively it takes relatively little time to define exception handling specifications for transactional workflow approaches. The main reason for this is that sequences of compensation activities can be dynamically generated for given exception types and states of progress of a workflow. Indeed when using transactional approaches there is no need to explicitly define sets of compensation activities needed for certain exceptions and states of a workflow. This can significantly reduce the time required to specify exception handling scenarios. Yet, it remains necessary to specify and enact suitable exception handling knowledge so that appropriate compensation activities can be dynamically generated. In the case of transactional workflow approaches 'spheres' are specified and deployed. The definition of spheres requires some time (i.e. the effort is classified as medium). Traditional workflow approaches have similar comparative properties to event-condition-action approaches with respect to effort, as indicated in Table 6.

Exception handling specifications developed using event-condition-action or traditional workflow modelling constructs and mechanisms are relatively inflexible when used to encode a change to the order of activity execution. In general changes to the order of activity execution, such as activity deletion or insertion or a precedence order change, will impact on the number & kind of states a workflow may enter and therefore on the set of compensation activities needed. As event-condition-action workflow approaches explicitly define states and sets of compensation activities and many such specifications may need to be modified[35]. On the contrary transactional workflow approaches do not explicitly model states and sequences of compensation activity, and are therefore less vulnerable to changes to the order of activity execution. Yet activity deletions or insertions or precedence order change may still require existing sphere specifications to be revised, but this may be done with relatively little effort. However it must be noted that although transactional workflow approaches were intended to be flexible they require a significant body of configuration information to be specified (i.e. to specify the attributes of spheres). Further they impose a general problem decomposition that may constrain modelling in a specific application domain

---

[35] Traditional workflow approaches do not explicitly model state but do model explicit sequences of compensation activities.

5.3 Correctness

It may be evident to the reader that exception handling specifications based on event-condition-action workflow approaches become very large and complex. Because of the many exceptions, workflow states and necessary compensation activities needed it is relatively easy to make a mistake. Moreover if a change occurs to the standard order of execution many rules associated with the exception handling specification have to be checked. Although the number of exceptions and workflow states is naturally limited for traditional workflow approaches (i.e. the model is only capable of handling logical activity failures), it is still necessary to embed explicit sequences of compensation activities in the workflow specification and mistakes can be made.

On the other hand transactional workflow approaches require some form of exception handling knowledge to be specified (i.e. via spheres), rather than by specifying workflow states and explicit sequences of compensation activities. Therefore it remains possible for workflow designers to make mistakes, albeit that the reduced size of the specification makes it easier to check. Also when the standard order of execution changes such specifications are more readily checked than would be many equivalent event-condition-action rules.

7. A new workflow approach for use in sales order processing domains

1.0 Introduction

In this chapter a new workflow approach for use in sales order processing domains is presented together with the set of design decisions that have been made.

2.0 Design decisions

In section 5 of chapter 6 the capabilities and limitations of three classes of workflow approach were considered from requirements viewpoints of both sales order processors (that equate mainly to requirements of completeness) and workflow system developers (that equate mainly to requirements of effort & flexibility and correctness). It was concluded that event-condition-action and traditional workflow approaches do not very well support the requirements of workflow system developers, whilst the needs of sales order processors are not very well addressed by transactional workflow approaches. Even though it may seem that an obvious solution would therefore be to achieve a combined use of the two classes of approach in a new way, in practice these approaches are rather different in nature and are difficult to combine. This conclusion may be clear from a consideration of differences in scope and emphasis of each class of workflow approach. For example, event-condition-action workflow approaches are focused on issues like event (A) type, (B) condition, and (C) action. Whereas for traditional workflow approaches two main concerns are (A) data flow modelling and (B) control flow modelling.

It is the author's belief that it is necessary to understand well characteristics of workflow approaches from perspectives of 'state' and 'transformation'. On the one hand a workflow approach must be able to reach relevant states to be able to handle exceptions of concern to sales order processors, whilst on the other hand a workflow approach should minimise the number of states that a workflow modeller has to define as over-complexity will impact in a negative way on criteria like effort, flexibility and correctness. Similar reasoning applies for rules that specify state transformations. Whilst traditional and event-condition-action workflow approaches both require the workflow states and transformation of sales orders to be pre-specified by a workflow modeller at design time (i.e. the designer is required to develop a static specification of compensation activities), transactional workflow approaches recognise that when an exception occurs the state of a workflow is related to the state of those activities that have already been performed. Transactional workflow approaches do not require exceptional states to be completely pre-specified[36]. These approaches view a workflow as a set of activities for which their results are subject to modification when exceptions occur. When an exception occurs the execution of a workflow is halted, its current results are amended with reference to properties of the exception, where after the workflow may continue its execution. Whilst the standard workflow is pre-programmed and some form of exception handling is specified, the runtime compensation activities that amend the current workflow are generated dynamically. Thus this class of workflow approach does not require all states that a workflow can enter to be explicitly modelled for the exceptions (types and instances) that may occur. In theory at least, this class of approach has a great potential as it should exhibit good performance measures with respect to effort, flexibility and correctness.

However to date transactional workflow approaches impose limits on the number of states that can be reached in support of runtime efforts of sales order processors. Typically only very constrained sales order processing application scenarios can be supported. Major limitations are that (1) the workflow does not recognise, nor possess capabilities to handle all types of exceptions that may occur in the domain of sales order processing, (2) they require the way that activities should be compensated to be pre-specified at the time of design (i.e. that either an activity should be updated or be cancelled) and (3) they assume that only one exception occurs at a time and that its propagated effects can be made permanent immediately.

Despite these evident limitations this research study has decided to adopt transactional workflow approaches as its base approach and to seek to extend the capabilities of this class of approach in a number of fundamental ways that cater for common sales order processing domain requirements.

---

[36] Although the transactional workflow approach Partial Rollbacks itself requires time to develop, modify and check for correctness.

3.0 A new workflow approach

A central concept, conceived and developed in order to produce the enhanced workflow approach, is that of 'workflow state and transformation'. The design hypothesis made and later tested is that workflow 'state' can be adequately captured by means of standard and exception workflows, whilst 'transformation' can be defined by a set of rules.

3.1 Workflow state: standard and exception workflows

It was assumed that a standard workflow can capture the 'base state' of a workflow, i.e. ignoring the possibility that exceptions might occur. Whilst it was also assumed that exception workflows specify alternative workflow states needed to handle given exceptions and decide upon suitable compensation actions. The design concepts are illustrated graphically in Figure 57.

Firstly it is important to note that a standard workflow specification should define all possible states that a workflow can enter, whilst a standard workflow instance captures a specific workflow state from all possible states[37]. In this study a standard workflow specification was also assumed to consist of a control flow and a data flow specification. A data flow specification will define the kind of data that may flow from one activity to another activity. This notion is based directly on concepts that underpin the IDEF0/SADT approach. A data flow specification defines the first basic order in which activities should be executed, bearing in mind that data consumer activities can only be executed after a data producer activity has executed. A control flow specification activates a sub set of these activities in a certain order. It was decided that control flow specifications should be represented by selected sets of activity, precedence link and junction constructs derived from IDEF3 and CIMOSA control flow modelling techniques. Thereby a control flow specification can define an order which can be put onto a data flow specification. It was also decided that a workflow instance should consist of a control flow instance and a data flow instance. The control flow instance should capture the state of activities, precedence links and junctions that have been triggered or evaluated. Thereby it should define those execution paths that have actually been enabled and those that are disabled. Whilst a data flow instance can capture the actual output of various activities.

---

[37] Frequently only the word workflow is used and the qualifier type or instance is left out. Then the user is required to derive from the context of the paragraph and section whether reference to a workflow type or a workflow instance is meant.

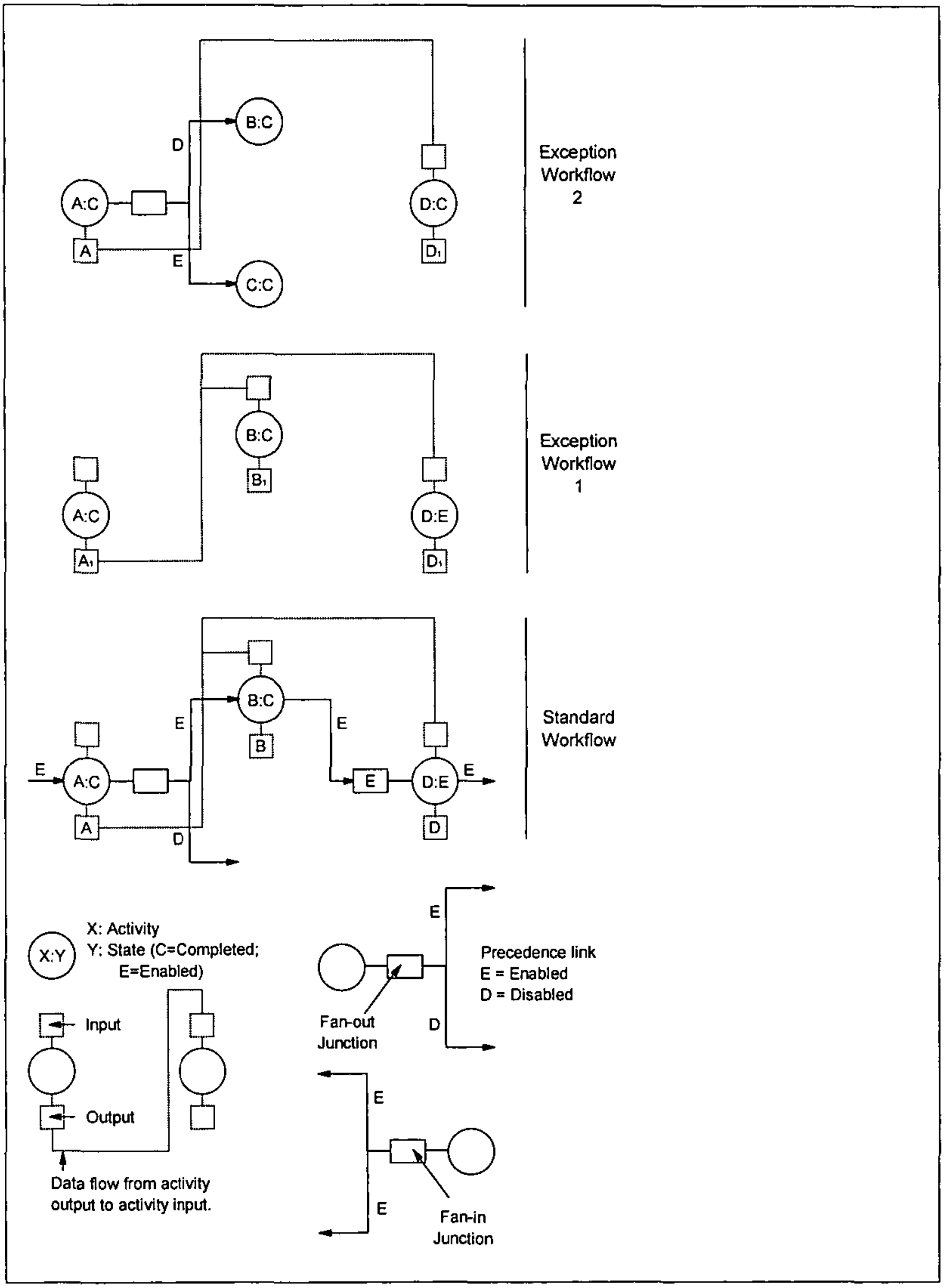


Figure 57: Single standard and multiple exception workflows.

Exception workflows are to be used to capture proposed changes to the standard workflow. They may encode properties of paths that have become invalid, new paths of execution, and activities that must be updated on still valid paths of execution for exceptions that have occurred. Thus it was decided that exception workflows would capture changes to sub sets of a standard workflow. A further decision was made that their design should also be based on the use of IDEF3, CIMOSA and SADT/IDEF0 modelling constructs, albeit that some construct extensions were found to be needed to capture all common kinds of change in the sales order processing domain.

To understand the contribution of this study it is necessary to understand the approach adopted to interpreting 'workflow states' especially with respect to exceptions. Transactional workflow approaches are based on a main workflow, yet changes to the main flow are applied immediately when an exception occurs. In the proposed approach, changes to the main flow are first stored with reference to exception workflows and can still be adjusted in certain cases where circumstances change. As a consequence it is possible to concurrently evaluate suitable responses to multiple exceptions. Indeed the new ideas constitute a significant extension to previous transactional workflow approaches. Because traditional and event-condition-action workflow approaches require all states of a workflow, including exceptional states, to be pre-defined then these approaches have not been selected as viable sales order processing workflow approaches, as discussed in the previous section. Essentially, with reference to the structure of the existing transactional workflow approaches improvements are twofold. Firstly the use of IDEF3 or CIMOSA control flow modelling constructs is enabled as these constructs can be more effective and are more mature (e.g. transactional workflow approaches do not include a junction notion). Although the flow of data is recognised in Partial Rollbacks, it is not modelled. In this respect use of SADT/IDEF0 modelling constructs is also enabled.

3.2 Workflow transformation

It was decided that standard and exception workflow transformations could be achieved in distinctive ways. Standard workflows would conform to IDEF3 and CIMOSA control flow logic; if an activity completes, the state of precedence links and junctions (as defined by the standard workflow) are re-evaluated and enabled activities may be started. When all activities have been executed successfully the execution of a standard workflow will be assumed to be complete.

However it was observed that the process of transforming exception workflows would need to be more complex but could be achieved with reference to a specified set of rules. Firstly, if an exception occurs an exception workflow can be started that will impact on the standard workflow, i.e. for those activities that have actually been performed, compensation activity is initiated. By such means exception workflows can impact coherently on standard workflows that have reached some way towards completion. This is because at execution time the new workflow approach is cognisant of those activities in a standard workflow that have actually been performed. It was decided that exception workflows should be generated dynamically from a knowledge of (A) properties of the exception that has occurred, (B) the actual state of the standard workflow and (C) all theoretically possible states of a standard workflow as defined by its workflow specification. According to the philosophy behind the new approach any current standard workflow will adopt one of the many possible states defined by its standard workflow specification. The actual state reached will depend upon the actual progress of the standard workflow when the exception occurs.

Furthermore, it was observed that exception workflows themselves may need to be updated or be cancelled. Since changes to the standard workflow are stored in exception workflows, it remains possible to reverse the impact of intermediate results should conditions change or should the cost of handling exception exceed some economic figure. In this way it is possible to evaluate the impact of exceptions acting on a standard workflow. Once an exception is accepted results stored in an exception workflow can be incorporated into the standard workflow, thereafter the state of other exception workflows can be updated.

It is the belief of the author that the use of the spheres concept (as proposed by ConTracts and Partial Rollbacks) to generate compensation activities is not appropriate for sales order processing. Although it is possible to define ways of compensating previously performed activities (e.g. by assigning activities to spheres and by overlapping spheres so as to create logical units of work) the sphere concept (A) cannot detect invalid and new paths of execution because of exceptions to the flow of control cannot be detected and (B) is not particularly suited to the identification of activities that must be updated because of exceptions to the flow of data. Basically the sphere concept ignores the fact that data and control flows provide 'binding' for sales order processing activities.

4.0 Workflow specifications

Hence it was determined that the new workflow approach will adopt the proven notion that the execution of activities in a workflow can be ordered by a control and a data flow specification. But in the case of the new approach the control flow specification should determine (A) which activities are eligible for execution, for cases where everything goes to plan and (B) which activities are to be updated, cancelled and blocked, in cases where an exception occurs. Also in the new approach knowledge contained in a data flow specification will be of relevance when an exception occurs as it will enable determination of those activities that should be updated.

Control flow specifications in the new approach will also capture business rules that trigger the execution of workflow activities by means of precedence links and junctions. The purpose of this design decision was twofold. Firstly, a control flow specification can serve to determine which activities are to be performed next (i.e. which paths of execution in a workflow are selected) when some activity completes. Secondly, it can be used to determine which business rules are no longer valid when an exception occurs and therefore those activities that should be cancelled (i.e. which paths of execution in a workflow are no longer valid). Therefore this study adopted a different view on the nature of a control flow specification to that adopted by CIMOSA, IEM and IDEF3; because these approaches view a control flow specification from the perspective of needing to execute parallel and sequential activities.

Consider for example the sales order control flow specification illustrated by Figure 58. At the base of this specification are two assumptions about a particular application scenario: (1) if the order quantity is smaller than 100 items, then these items should be delivered directly from the company's own warehouse, otherwise the company's preferred supplier should deliver the goods. The former requires the following activities to be performed: (A) 'maintain sales order properties', (B) 'allocate stock', (C) 'send order confirmation' and (D) 'manage order delivery schedule'. The latter requires activities (E) 'check creditworthiness customer', (C) 'send order confirmation' and (F) 'generate purchase order' to be performed. Yet the company itself will remain responsible for invoicing the customer. The same control flow specification also dictates that in case of an exception some activities may need to be cancelled and others be started. Consider for example the situation where the order quantity is 50 and activities A, B and C have been performed. Now assume that an exception occurs and the order quantity is changed to 200. In such a case the control flow specification dictates that activities B and C should be cancelled, following which activity E should be started. A traditional perspective would define the control flow specification as follows. After activity A has been performed a fan-out junction splits the single path of execution into multiple parallel paths of execution. While activity set B, C and D and activity set E, C and F are to be performed sequentially, any activity from these sets can be performed in parallel. These paths of execution are merged by a fan-in junction before activity G can be performed. Then a workflow can complete.

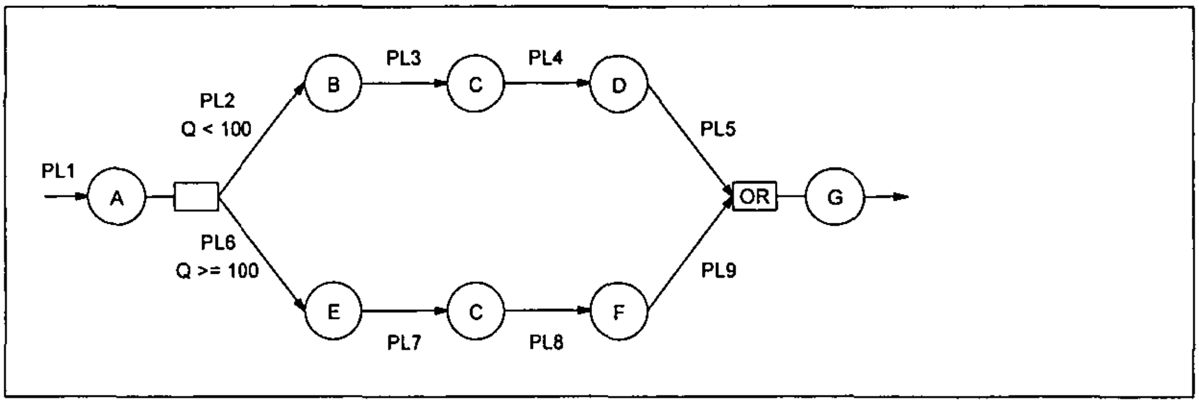


Figure 58: Example sales order control flow specification.

In the new workflow approach it was decided that a data flow specification would serve to determine those activities that may need to update their output in cases where either change in external information change occurs or the execution of an activity fails. Therefore the data flow specification would help determine those downstream activities that would be affected by a change to the output of an upstream activity, or alternatively, in cases where execution of an activity fails, those upstream activities that may need to modify their output. For example if a change occurs in the output of activity A (i.e. 'Maintain sales order properties') then inputs to many downstream activities (such as activity B, C and D) may require modification. Also if insufficient stock is allocated to an order (so that downstream activity B fails) the output of upstream activity A ('Maintain sales order properties') or the external information E2 may require modification.

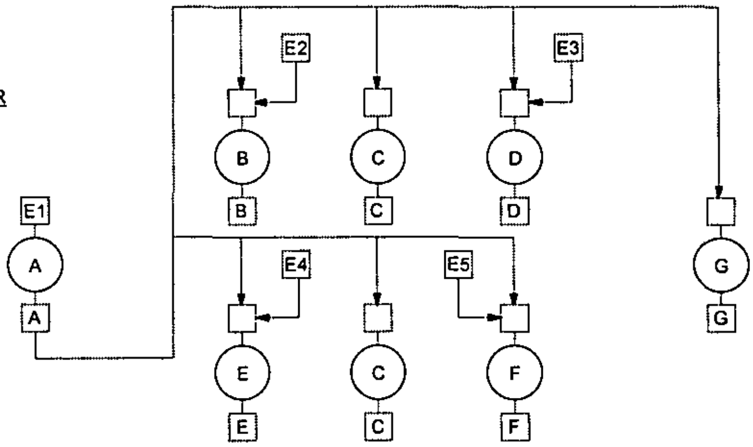
E1:
10 SALES ORDER NUMBER
20 CUSTOMER
30 ITEM
40 DESCRIPTION
50 QUANTITY
60 DUE DATE
70 PRICE
80 AMOUNT

E5:
10 SUPPLIER
20 ITEM
30 PRICE

E2:
10 ITEM
20 WEEK
30 QUANTITY AVAILABLE

E3:
10 TRUCK
20 LOADING VOLUME

E4:
10 SALES ORDER NUMBER
20 CUSTOMER
70 AMOUNT
80 INVOICED



OUTPUT A:
10 SALES ORDER NUMBER
20 CUSTOMER
30 ITEM
40 DESCRIPTION
50 QUANTITY
60 DUE DATE
70 PRICE
80 AMOUNT

OUTPUT B:
10 ITEM
20 WEEK
30 ORDER NUMBER
40 QUANTITY ALLOCATED

OUTPUT C:
10 SALES ORDER NUMBER
20 CUSTOMER
30 ITEM
40 DESCRIPTION
50 QUANTITY
60 DUE DATE
70 PRICE
80 AMOUNT

OUTPUT D:
10 DELIVERY NO
20 TRUCK
30 DATE
40 ORDER NUMBER

OUTPUT E:
10 CUSTOMER
20 CREDITWORHTY
30 OUTST' PAYMENTS

OUTPUT F:
10 PURCHASE ORDER NUMBER
20 ITEM
30 QUANTITY
40 DUE DATE

OUTPUT G:
10 SALES ORDER NUMBER
20 AMOUNT PAYED

Figure 59: Example sales order data flow specification.

It was decided that a data flow specification should define the flow of data amongst activities by means of data use & produce definitions and data flow connectors. The data use of an activity specifies the kind of data an activity requires to perform its operation, and the data produce defines the kind of data an activity generates. Data use and produce are defined in terms of data element types (or record types in terms of relational data structures) and their respective data item types (or attribute types in terms of relational data structures). The purpose of data connectors is to define the data dependencies amongst activities by linking the data produce by upstream activities to the data use of downstream activities. If a logical activity uses data from an another activity it is termed a consumer or downstream activity, and if an activity generates data for activities it is a producer or upstream activity. The data flow specification used by the new workflow approach conforms to a SADT/IDEF0 data flow specification. However some of the modelling notations have been adapted with the aim that instances of data flows can be used in combination with instances of control flow specifications.

Consider for example the data flow specification from Figure 59. The data produce of activity B consists of four record attributes 'ITEM', 'WEEK', 'ORDER NUMBER' and 'QUANTITY ALLOCATED', etc. It's data use comprises external information E2 and the output of activity A. A data connector connects the data produce of activity A with the data use of logical activities B, C, D, E, F and G.

5.0 Workflow state

5.1 Standard workflow

In the new workflow approach a standard workflow will be uniquely defined by (1) the state of its data flow, (2) the state of its control flow and (3) the state of its activities. A data flow instance will consist, for as far as the workflow has currently progressed, of a set of activities, their inputs and outputs, along with data dependencies that link activity outputs to activity inputs. It was observed that activity outputs can be encoded via *sets of record attribute values* such as strings, integers, dates, etc. which can take on different values over time[38]. A control flow instance consists of the state of the business rules (i.e. precedence links and junctions) that have been evaluated in order to reach the current workflow state. It was observed that the necessary states of associated business rule could be either *enabled*, *disabled* or *undefined*. Further observed was that the various states that a 'standard activity' can reach are: *enabled*, *started*, *completed*, *logical failure* and *blocked* (where the semantics of these states are defined in Table 7). The status of each path of a control flow would then be determined by analysis of relevant activity states (i.e. enabled, started, completed or blocked) and the states of relevant precedence links and fan-in junctions.

Valid paths of execution are defined by evaluated precedence links and fan-in junctions for which the state is *enabled*, and by activities whose state is either *enabled*, *started*, or *completed*. Invalid paths of execution are defined by evaluated precedence links and fan-in junctions for which the

---

[38] N.B. The input of an activity is not captured as it is readily available as the output of other activities.

state is *disabled.* Consider for example the control flow instance from Figure 60. This control flow instance has one valid path and one invalid path. The valid path consists of activities A, B, C, D and G, precedence links PL1, PL2, PL3, PL4 and PL5, and a fan-in junction. The state of activity G is *enabled* since it is at the front of a path. The invalid path consist of the single precedence link PL6.

Activity state	**Semantics**
1. ENABLED (E)	This state signifies that an activity remains to be performed.
2. STARTED (S)	This state signifies that an activity is being performed and some outputs may have been generated. For some reasons end users may be temporarily interrupted whilst doing their work and need to be reminded that the activity still needs to be completed.
3. COMPLETED (C)	This state signifies that an activity has been successfully performed and all necessary outputs have been generated.. At this stage no further processing is required.
4. LOGICAL FAILURE (F)	This state signifies that the execution of an activity has failed and that either (A) the execution of the activity must be re-continued at some point in time or that (B) the input to the activity needs to be changed.
5. BLOCKED (B)	This state signifies that an activity may temporarily not be performed and so far no outputs have been generated. In a situation where an exception has occurred it may not be certain whether the activity needs to be performed or if the input to the activity needs changing.

Table 7: States of a standard activity.

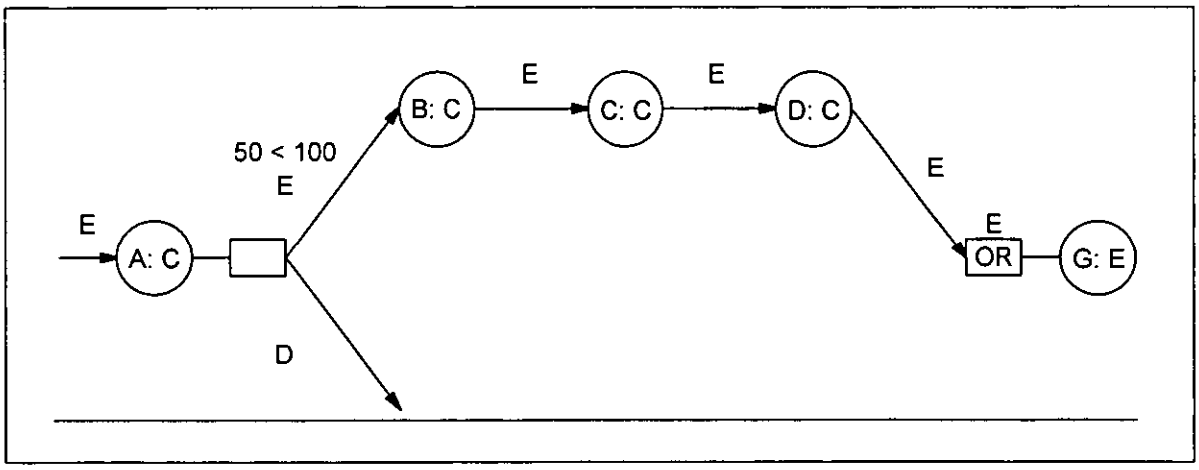
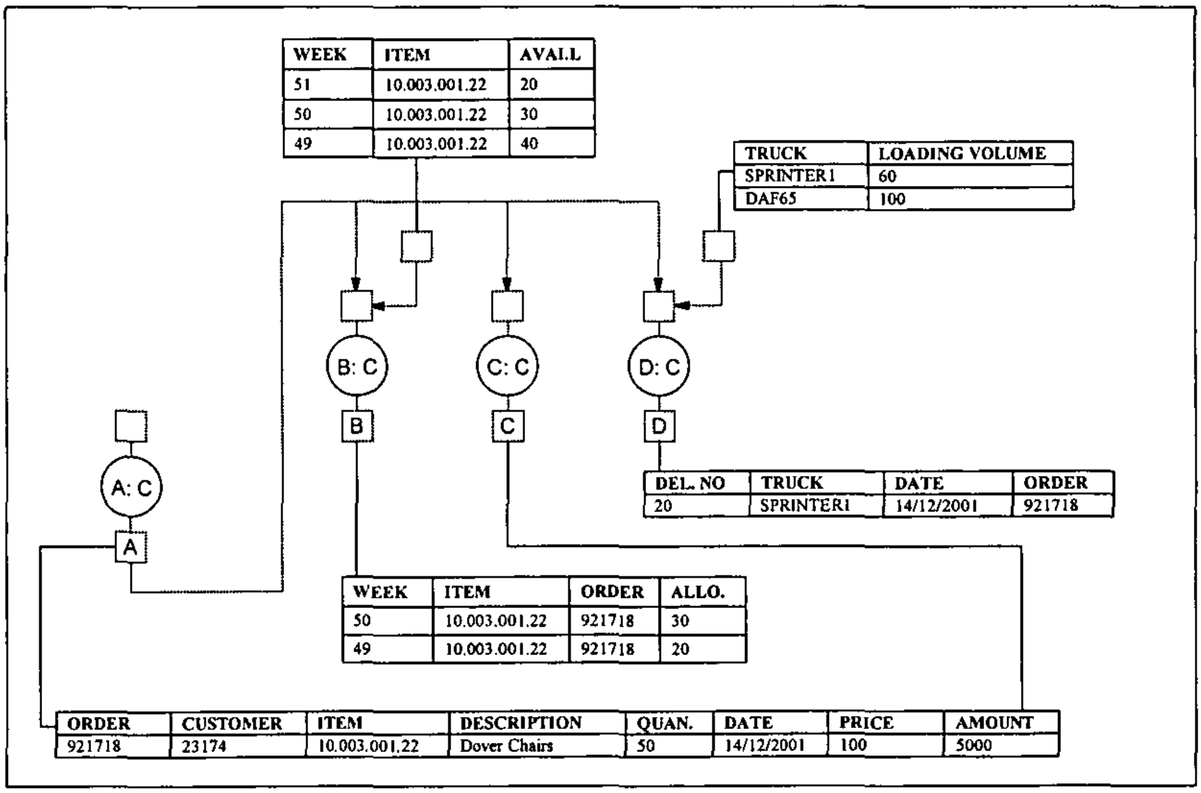


Figure 60: State of a control flow instance.

Consider for example the state of the data flow instance shown in Figure 61 respectively. Assume that activity A (i.e. 'Maintain Sales Order Properties') simply stores the order properties as made available by the customer. Customer 23174 orders 50 'Dover' chairs (i.e. item 10.003.001.22) to be delivered on 14/12/2001. Sales order information generated by this activity is used by three downstream activities. Firstly activity B ('Allocate Stock') assigns relevant chair components from week 49 and 50 to this order (i.e. 30 and 20 items respectively). Ideally only items for week 50 would have been allocated (in order to minimise stock holding costs) yet in this week only 30 items remain available. Therefore 20 items from an earlier week require selection in order to make the chairs in week 50. Secondly activity C ('Send Order Confirmation') simply prints the sales order information, providing a paper copy. Thirdly activity D ('Manage delivery schedule') plans a van delivery (i.e. SPRINTER1) related to this order on day 14/12/2001.



WEEK	ITEM	AVAI.L
51	10.003.001.22	20
50	10.003.001.22	30
49	10.003.001.22	40

TRUCK	LOADING VOLUME
SPRINTER1	60
DAF65	100

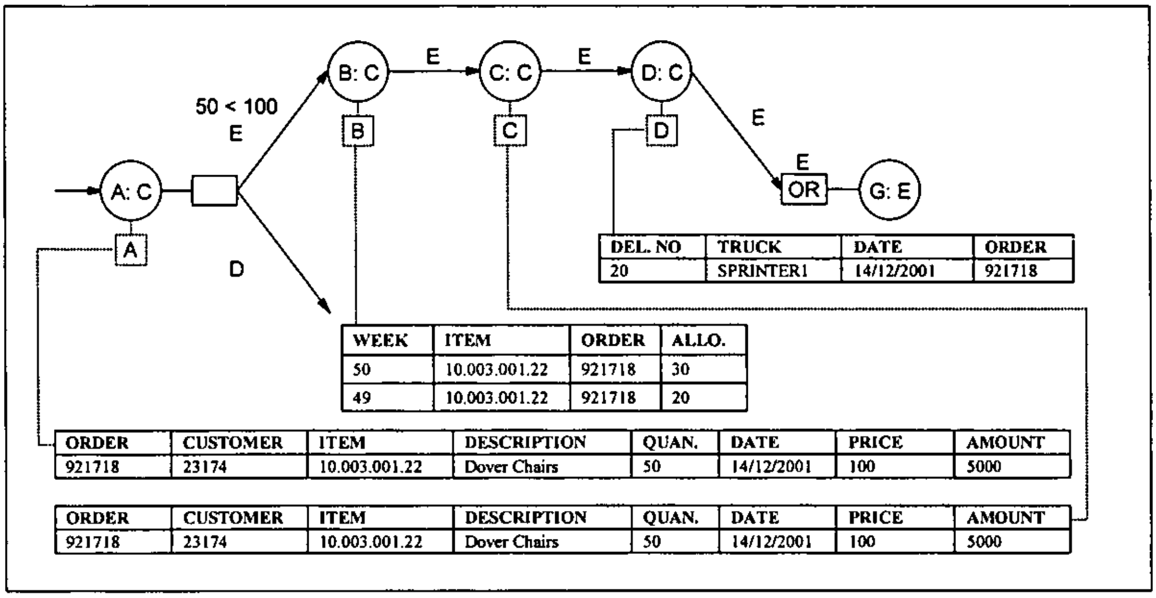
DEL. NO	TRUCK	DATE	ORDER
20	SPRINTER1	14/12/2001	921718

WEEK	ITEM	ORDER	ALLO.
50	10.003.001.22	921718	30
49	10.003.001.22	921718	20

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	50	14/12/2001	100	5000

Figure 61: Example data flow instance.

Although the state of a workflow instance can be represented by simply overlapping graphical representations of its data flow instance and its control flow instance, many interrelationships exist between workflow modelling constructs and will clutter the view. Therefore to simplify matters in the new workflow approach a workflow instance will be represented graphically by its control flow instance and associated activity outputs (as shown in Figure 62). Data dependencies will not be shown graphically when representing a workflow instance as these dependencies will be coded in the data flow specification.



DEL. NO	TRUCK	DATE	ORDER
20	SPRINTER1	14/12/2001	921718

WEEK	ITEM	ORDER	ALLO.
50	10.003.001.22	921718	30
49	10.003.001.22	921718	20

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	50	14/12/2001	100	5000

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	50	14/12/2001	100	5000

Figure 62: Example representation of a standard workflow instance.

5.2 Exception workflows

In the new workflow approach an exception workflow will record any changes that need to be made to information plans generated by sales order activities in order to handle an exception. An exception workflow consists of a set of interrelated exceptions and solutions. A root exception is an exception that triggers an exception workflow, whilst a derived exception is an exception that has been directly triggered by a solution (in order to handle another exception). An exception always requires a solution and a solution may result in nil, one or multiple exceptions.

Earlier thesis analysis showed that in sales order processing domains a root exception is either a logical activity failure or an external information change that triggers the exception workflow. A logical activity failure corresponds to an exception where a workflow activity cannot complete its execution successfully given its input and the conditions it has to meet (even though it may be able to convert part of the input to a useful output). For example if a SOP activity updates a plan for a sales order it may happen that this new sales order cannot be accommodated because of already planned orders. Typical examples of such situations are 'cannot allocate sufficient stock to order', 'delivery schedule not optimal', 'customer not creditworthy', etc. From a technical perspective a logical failure indicates that an activity cannot convert its input to a required output taking into account pre-defined predicates. Thus if an activity is to complete successfully the next time its runs, its input has to be different (or its completion conditions may need to be relaxed). An external information change is an exception where some information provided by the 'world' outside the sales order processing system is modified. Typical examples of such changes are sales order changes requested by the customer and master production schedule changes requested by other organisational units from the same company. Usually such a change is formulated in the form of a request for change rather than the change being imposed on the sales order processing system.

A solution is a set of corrections to sales orders and information plans needed to handle an exception that has occurred during some activity execution. From a technical perspective a solution consists of some changes to the data use (also termed input) and/or data produced (also termed output) so that the new input will be consistent with the new output according to the logic of the activity. Consider for example a case where insufficient stock can be allocated to a sales order (i.e. the execution of an activity fails). A number of solutions can be suggested by a compensation activity attached to this problem. Firstly it may suggest some changes to the master production schedule so that sufficient stock will be available in the weeks required (i.e. changes to the data use). Secondly it may propose later due dates for the sales order (i.e. also changes to the data use). Thirdly it may suggest changes to items already allocated to other orders (i.e. changes to the data produced) and possibly to the due dates of orders (i.e. changes to the data use). Thus a combination of solution types may be adopted. Another example of changes to the output is the case where some extra items are ordered at a very late stage. In such a case a compensation activity can calculate any necessary changes to the delivery schedule so the extra items can still be delivered on time.

Compensation actions taken with respect to previously performed activities takes the form of adapting their inputs or outputs so that an appropriate response is made when an exception occurs. Such an exception may (1) require the output of a completed activity to be modified so its value is matched to its new input value, or alternatively, its input value to be modified so that this corresponds to its new output value[39] or (2) require its output to be fully undone. In the former case the activity should be 'updated' while in the latter case it should be 'cancelled', as defined by Table 8. This design has resulted from the fact that both (A) upstream activities can propose new input for downstream activities and (B) downstream activities can propose new output for upstream activities (as discussed in chapter 5). Thus the flow of data is truly bi-directional for sets of producer/consumer activity while traditionally the flow of data has been considered unidirectional.

---

[39] Traditionally update compensation activities only update part of the original output for some changes to the input.

Type	Definition
Update	Update part of the output (or input) in response to changes to the input (or output).
Cancel	Remove the complete output of an activity.

Table 8: Types of compensation activities and their effect.

Consider for example the exception workflow illustrated by Figure 63. This exception workflow may have been triggered by a request from the customer to increase the ordered quantity for item 10.003.001.22 by 25 (as defined by root exception E'). After compensation activity A ('Modify Sales Order Properties') has processed this exception, its output is modified (as defined by solution A'). As a consequence three derived exceptions (A') are propagated to activities 'Allocate Stock', 'Send Order Confirmation' and 'Maintain Delivery Schedule' respectively. As a consequence the output of these three activities requires adjustment. Firstly 25 extra items need to be allocated to order 921718. In week 49 only 10 items are available so 15 items from week 48 will need to be allocated. Secondly a new order confirmation must be sent. Thirdly it is necessary to change the delivery schedule. As now 75 items need to be delivered, it may be decided that it is better to deliver them a day later with a larger truck (i.e. DAF65).

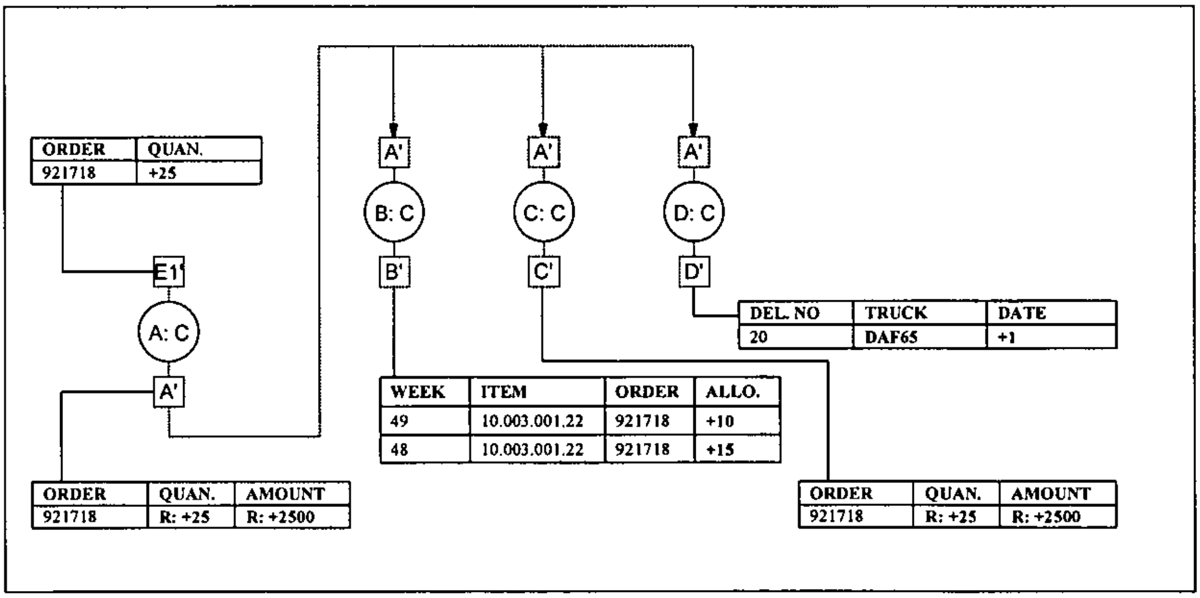


Figure 63: Example exception workflow.

6.0 Handling exceptions

A new approach to handling exceptions was conceived bearing in mind the observed nature of interactions within a network of sales order processing exceptions that comprises workflow specifications and workflow instances and their states. As exceptions occur, the aim of this new approach is to specify and enact appropriate data flows and control flows within two distinct phases as follows:

1. An evaluation phase. During this phase, when an exception occurs the objective is to determine:

    - those paths of execution that have become invalid, i.e. paths along which business rules have become invalid and for which activity outputs are no longer required;
    - those new paths of execution that may be executed, i.e. paths along which business rules have become valid and along which activities may be started;
    - those activity outputs that remain on paths of execution but have become partly (or completely) invalid; and
    - those activities that should be blocked.

    Having determined information related to the above issues, all changes that impact on a standard workflow (as a consequence of the occurrence of the root exception) are stored by attributing states and values to modelling constructs used to characterise the exception workflow.

2. A completion phase (which follows the evaluation phase). During the completion phase the objective is:

    - to make a decision as to whether to accept or reject the compensation plan as specified during the evaluation phase and coded up by the exception workflow instance.
    - dependant on this decision to apply changes specified by the exception workflow instance to the standard workflow instance and to make workflow changes permanent (if the exception has been accepted) or to simply discard the exception workflow (if the compensation plan is rejected). In both cases it was observed that blocked activities in a standard workflow should be released.

6.1 The evaluation phase

The evaluation phase follows the occurrence of an exception and for the new workflow approach during this phase:

1. an exception workflow is built by using knowledge about necessary changes to the standard workflow. This can be achieved in a stepwise manner starting from the activity at which the root exception occurred and progressing onto other activities affected by derived exceptions as they propagate through the standard workflow. It was determined that this can be achieved in the following way:
    - by simulating the effect of executing a compensation action on a standard workflow activity needed to deal with a root or a derived exception,
    - by determining derived exceptions from the effects of a previously compensation action by making reference to (A) the effects of a compensation activity and (B) data flow and control flow dependencies connecting activities in a workflow,
    - continuing to apply previously mentioned steps on a repetitive basis to enable building of a complete exception workflow.
2. the total cost of compensating for the root and derived exceptions can be estimated, using appropriate domain knowledge and rules.

6.1.1 Handling exceptions to the flow of data

Compensation activities handle exceptions by generating a suitable compensation solution. As discussed in chapter 5 an exception may render the output of an activity partly invalid or it may require the input of the activity to be revised (and this can affect the output of upstream activities or validity of external information). Compensation activity can either take the form of (A) execution of a software algorithm that automatically enacts necessary changes or (B) a software mechanism that requests human sales order processors to enter needed changes. An example of an algorithmic compensation activity might follow the occurrence of an exception of activity 'de-allocate stock' because an order is cancelled. Such a compensating software algorithm might automatically de-allocate all items (corresponding to different MPS weeks) that were previously assigned to the order and making new stock available for allocation to other orders. An example of a software mechanism that requests human sales order processors to enter needed changes might follow an exception impacting on a 'modify delivery schedule' activity. Such a compensation mechanism may require input from a human sales order processor as to which truck should be used for an order and request a suitable delivery date, instead of calculating a delivery schedule by automatically taking into account existing deliveries. Thus compensation solutions may correspond to various generated compensation activities that result in coherence between data relationships connecting activities of a standard workflow.

Derived exceptions to up- and downstream activities can be generated by taking into account the flow of data between activities and the properties of proposed compensation solutions: This is explained in 1 and 2 below.

1. Changes to the output of an upstream producer activity (as specified in a compensation solution) can be mapped onto changes to the input of downstream consumer activities. First of all it is necessary to determine all possible consumer activities that will be impacted on by a proposed solution and their intended data use by referencing the data flow specification. Typically in practice sales order processing consumer activities will use sub sets of data generated by producer activities. Here different types of derived exception can be distinguished with respect to their use of three sets of data items, namely (A) data items generated at the output of an upstream activity, (B) data items used as input by a downstream activity and (C) changes to data items generated at the output of an upstream activity. Here it is observed that the set of changes to data items used as an input by any upstream activity can be determined by considering the overlap in the other three data sets (as illustrated in Figure 64).

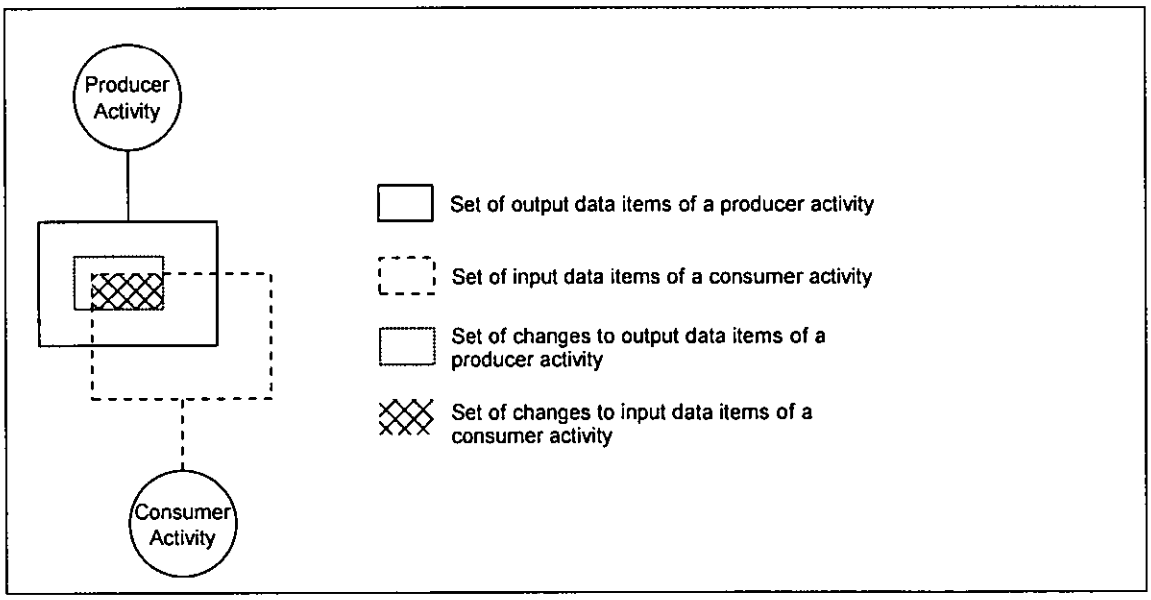
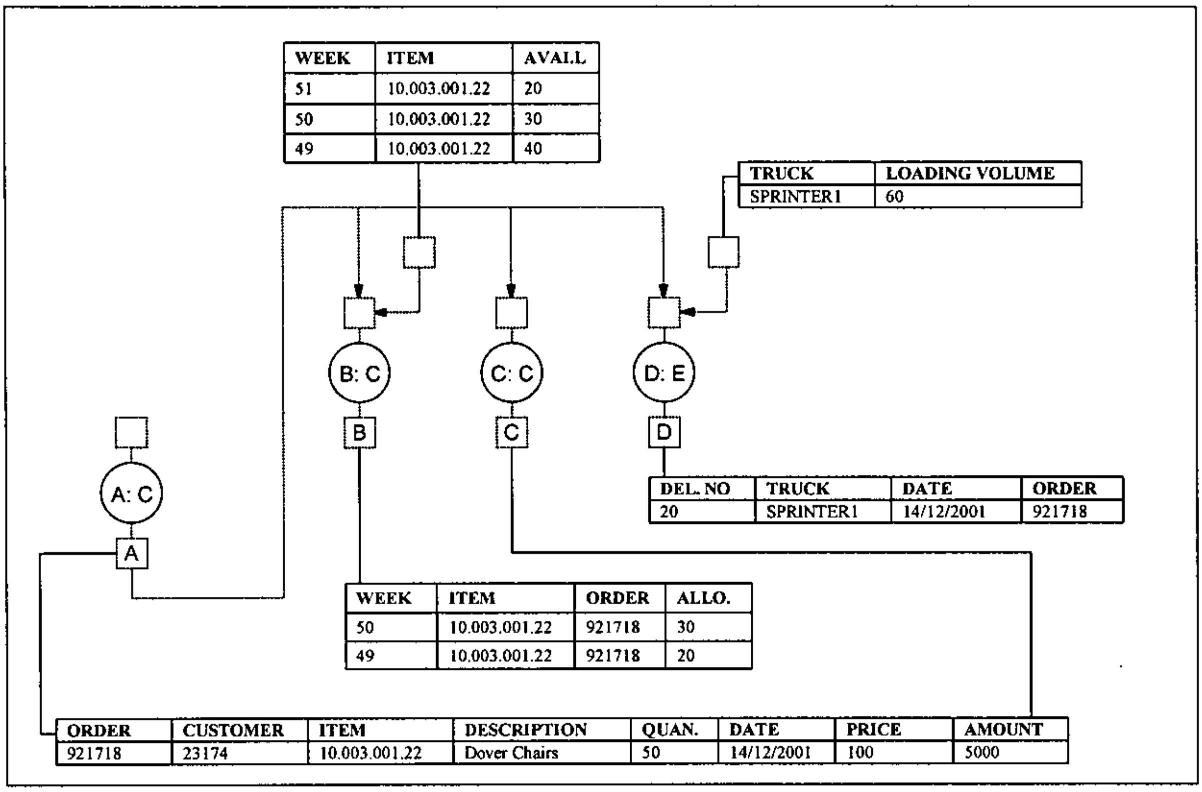

Producer Activity
Consumer Activity
Set of output data items of a producer activity
Set of input data items of a consumer activity
Set of changes to output data items of a producer activity
Set of changes to input data items of a consumer activity

Figure 64: Resultant set of changes to the input of a downstream activity.

2. Changes to the input of a downstream consumer activity (that occur as a consequence of enacting a compensation solution) can be mapped to changes to the output of upstream producer activities. Firstly the producer activity and the data it produces will be determined by properties of the data flow specification for a given consumer activity. As in sales order processing domains only records of a certain type are generated by a single producer activity, the process of determining this mapping is relatively straightforward.

Consider for example the data flow specification depicted by Figure 59 and the data flow instance shown in Figure 65. For example in this data flow instance so far four sales order processing activities have been performed on order 921718, i.e. activities A, B, C and D. Suppose now an exception workflow is triggered by a request from the customer to increase the ordered quantity for item 10.003.001.22 by 25 (this resulting in root exception E').



WEEK	ITEM	AVAIL
51	10.003.001.22	20
50	10.003.001.22	30
49	10.003.001.22	40

TRUCK	LOADING VOLUME
SPRINTER1	60

DEL. NO	TRUCK	DATE	ORDER
20	SPRINTER1	14/12/2001	921718

WEEK	ITEM	ORDER	ALLO.
50	10.003.001.22	921718	30
49	10.003.001.22	921718	20

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	50	14/12/2001	100	5000

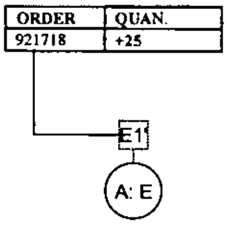
Figure 65: Example data flow instance for sales order processing.

When exception E' occurs compensation activity A: 'Modify Sales Order Properties' will be enabled ready for execution (as shown in Figure 66.A). As soon as this compensation activity is enacted, solution A' will capture changes to the output. In this example case, the variable QUANTITY will be increased in value by 25[40]. Consumer activities (B) 'Allocate Stock', (C) 'Send Order Confirmation', and (D) 'Maintain Delivery Schedule' can be identified by interpreting the data flow specification from Figure 59 for producer activity A. In the example scenario it can be observed that the inputs of all consumer activities will be affected as the attribute QUANTITY is specified in their data use function. As a consequence three derived exceptions (A') result and will impact upon activities 'Allocate Stock', 'Send Order Confirmation', and 'Maintain Delivery Schedule'. This is illustrated in Figure 66.B. It follows that the output of these activities must be adjusted (i.e. compensation solutions for derived exceptions

---

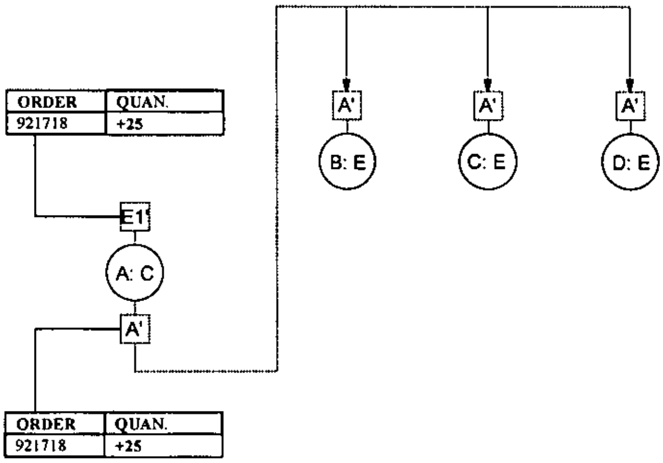
[40] As activity A is a simple data entry activity, the properties of the exception and solution will be the same.

now need to be determined). Firstly 25 extra items need to be allocated to order 921718. As in week 49 only 10 items remain available and 15 items from week 48 need to be allocated. Secondly a new order confirmation should be sent.



ORDER	QUAN.
921718	+25

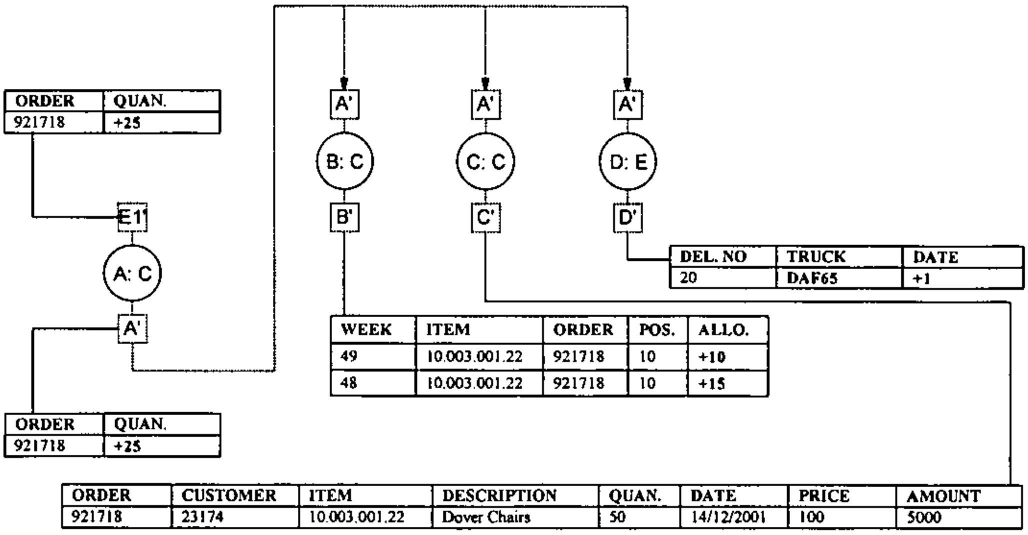
(A) State 1 of data flow instance.



ORDER	QUAN.
921718	+25

ORDER	QUAN.
921718	+25

(B) State 2 of data flow instance.



ORDER	QUAN.
921718	+25

DEL. NO	TRUCK	DATE
20	DAF65	+1

WEEK	ITEM	ORDER	POS.	ALLO.
49	10.003.001.22	921718	10	+10
48	10.003.001.22	921718	10	+15

ORDER	QUAN.
921718	+25

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	50	14/12/2001	100	5000

(C) State 5 of data flow instance.

Figure 66: Three example workflow states.

Thirdly it is necessary to change the delivery schedule. As now 75 items need to be delivered, it may be decided by a human sales order processor that it is better to deliver the items a day later with a larger truck (i.e. DAF65). The state of the exception data flow instance (built in response to exceptions in this example case) is shown in Figure 66.C.

6.1.2 Handling exceptions to the flow of control

Thus it was observed that a change to the output of an activity may change the state of business rules and that therefore (A) existing paths of execution may become invalid or (B) new paths of execution may need to be instantiated. The process of generating an exception workflow was understood to involve (1) identifying any necessary business rules for which predicates refer to outputs generated by activities impacted upon by exceptions, (2) determining whether the state of these predicates has changed and if this is the case (3) using information about predicate changes to identify associated changes to the validity of paths of execution. Also activities at the front of invalid paths of execution must be blocked as costs are likely to be incurred if these activities are performed and subsequently the exception path is cancelled.

Changes to the state of any business rule can be determined by comparing (1) the initial state of a business rule (which can be determined from the state of the standard workflow when the root exception occurred) and (2) the new state of the business rule (as determined by substituting values of modified variables in the predicate of the business rule and evaluating resultant expressions). By comparing the 'relative states' of business rules any state change can be detected. Here it was determined that four relative states of a business rule can occur, as illustrated by Table 9.

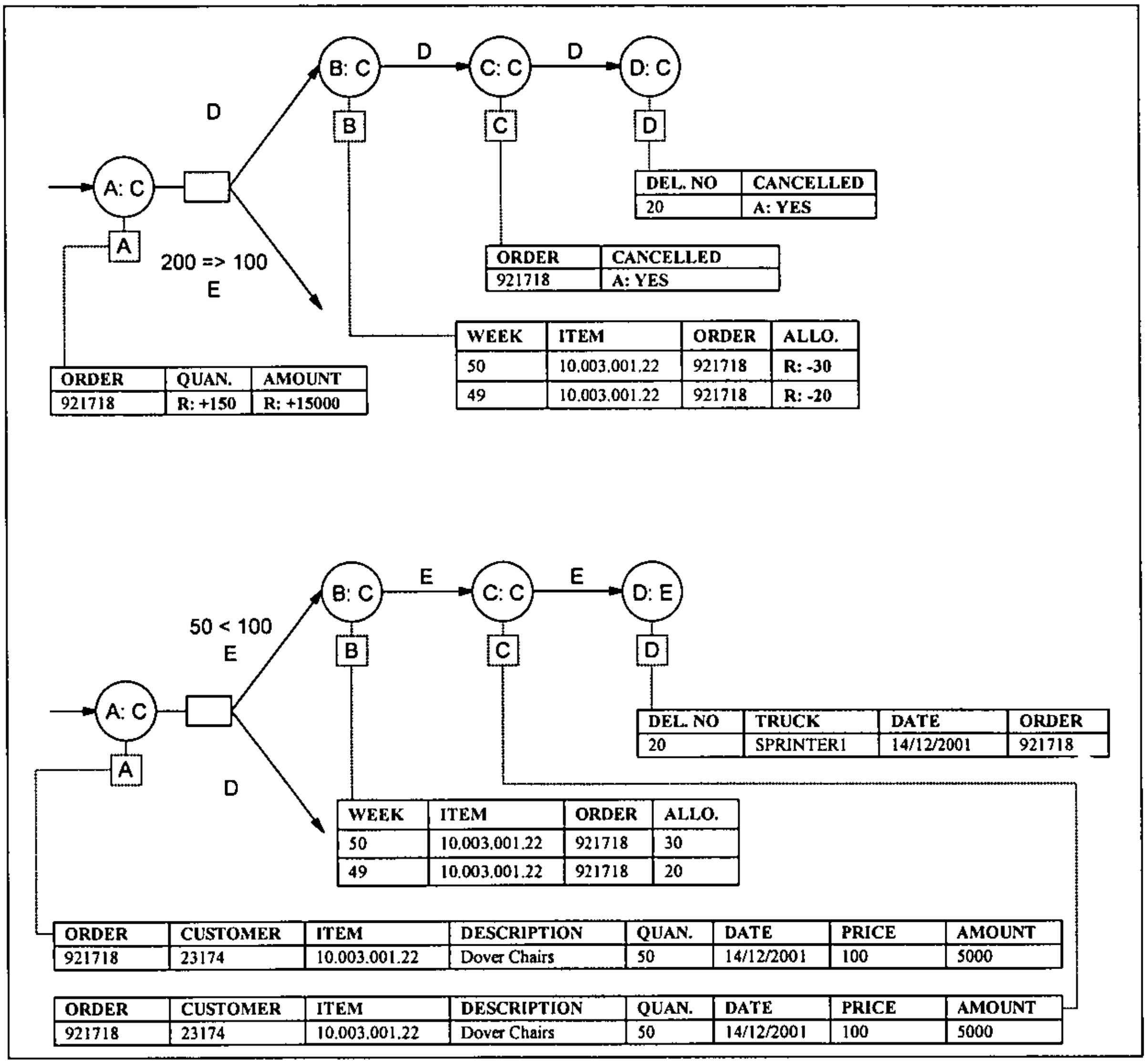
Case	Initial state	New state	Change	Relative state	Semantics
1	Enabled	Enabled	No	Still Enabled	Points to a still valid path of execution
2	Enabled	Disabled	Yes	Newly Disabled	Points to an invalid path of execution
3	Disabled	Enabled	Yes	Newly Enabled	Points to a new path of execution
4	Disabled	Disabled	No	Still Disabled	Points to a still invalid path of execution

Table 9: Possible relative states of business rules.

Only the second and third cases in Table 9 will result in a derived exception. In cases where the relative state of a business rule is 'Newly Enabled', a new path of execution needs to be entered and thus a derived exception will result. If the relative state of a business rule is 'Newly Disabled' and the business rule points to an activity that has already been executed then its output result should be cancelled But if it points to an activity that is enabled for execution or has been started, then activity execution should be blocked. In cases where a relevant activity has been completed, the state of any following business rule that has triggered should be re-evaluated. Thus by repeated use of this detection technique the occurrence and propagated effects of exceptions can be determined and recorded in support of subsequent steps in the analysis.

Consider again the control flow specification illustrated by Figure 58. Assume that the progress of the standard workflow reaches the state illustrated by Figure 67.B. This graphical representation of the standard workflow indicates that activities A, B, C and D have been successfully completed (as their current state is Completed) and that activity G is ready for execution (because its state is Enabled). Under such conditions assume that an exception occurs because of a change to the input of completed activity A.

In this example case execution of an update compensation activity for activity A could result in the output variable QUANTITY of activity A being updated to 200 (i.e. so that this variable is changed in value by +150). Since the predicate of precedence link PL2 and PL6 refers to variable QUANTITY its state will be re-evaluated following compensation of the root exception E. In this case it would be found that precedence link PL2 is no longer enabled. But according to the standard workflow description the precedence link PL2 has previously triggered execution of activity B. Hence the change in state of PL2 requires the output results of activity B to be cancelled. As activity B has previously enabled precedence link PL3, this precedence link is no longer valid and should be set to disabled. As a consequence activity C should be cancelled, precedence link 4 should be disabled, activity D should be cancelled and precedence link 5 should be disabled. Activity G does not require any compensating action to be performed because it had only been enabled and its execution had not been started. Therefore execution of activity G needs to be blocked temporarily.



DEL. NO	CANCELLED
20	A: YES

ORDER	CANCELLED
921718	A: YES

WEEK	ITEM	ORDER	ALLO.
50	10.003.001.22	921718	R: -30
49	10.003.001.22	921718	R: -20

ORDER	QUAN.	AMOUNT
921718	R: +150	R: +15000

DEL. NO	TRUCK	DATE	ORDER
20	SPRINTER1	14/12/2001	921718

WEEK	ITEM	ORDER	ALLO.
50	10.003.001.22	921718	30
49	10.003.001.22	921718	20

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	50	14/12/2001	100	5000

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	50	14/12/2001	100	5000

Figure 67: Example control flow exception workflow.

6.2 Completion phase

During the completion phase a decision is made either (A) to assimilate the exception workflow into the standard workflow, i.e. to accept the root exception and its derived exceptions and their recommended compensation solutions or (B) to discard the exception workflow and continue executing the standard workflow, i.e. to reject the root exception and its effects. In the case of (A) invalid paths of execution should be omitted from the standard workflow, locks on data items should be released, new paths of execution should be started and changes to activity outputs should be made permanent. In case of (B) any locks on data items should be released as well as any blocked activities on existing paths of execution in the standard workflow. In general it was observed that exception workflows should be processed in a top-down fashion, starting at the root exception and its associated compensation action (or solution), moving on to derived exceptions induced by this compensation action, and so on.

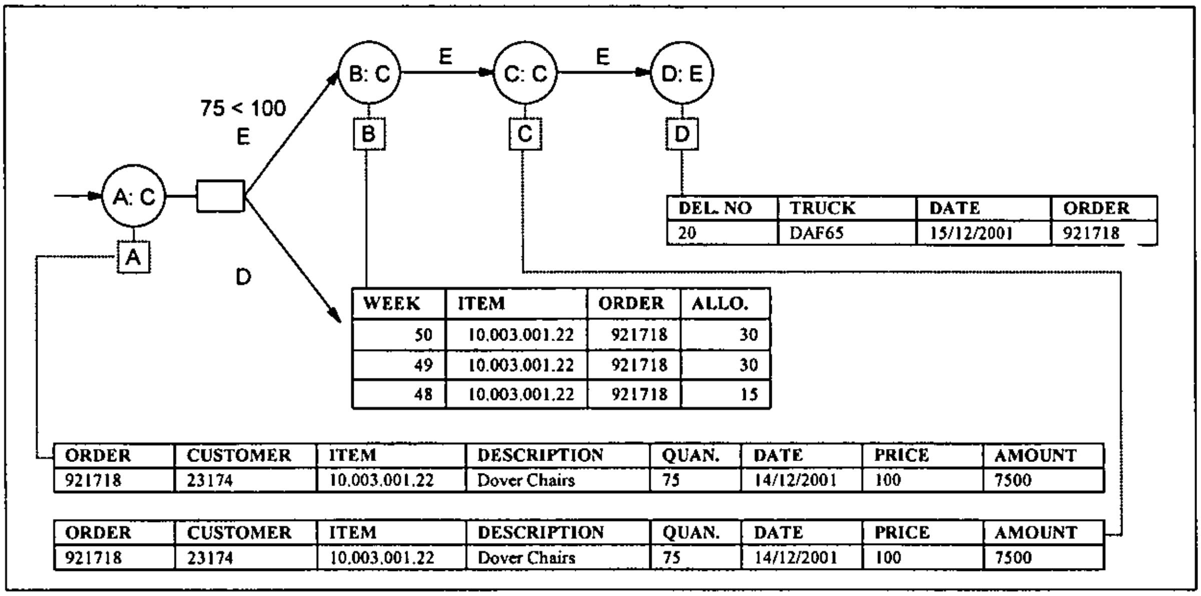
In situations where a root exception is accepted the following actions should be taken:

1. Needed changes should be made to the state of business rules. In the case where a business rule is newly enabled in the exception workflow, then the occurrence of this business rule in the standard workflow should be set to enabled and either (A) its target activity is enabled or (B) the state of its target fan-in junction should be (re-)evaluated. In the case where a business rule is newly disabled, the occurrence of the business rule in the standard workflow should be set to disabled and any following invalid business rules need to be simply removed from the standard workflow as should those activities that have been cancelled.

2. Needed changes to the output of standard activities (as generated by update or cancel compensation activities) should be made permanent. If a designated change to a data item is absolute then the new value should replace the old value, but if a designated change is relative the value of the data item it should be decreased or increased by the nominated quantity.

3. Needed changes to the input of standard activities should be simply discarded (as they are not stored in the standard workflow) and locks on data items should be released.

Assume that in the example case illustrated by Figure 66.C the root exception E' and its changes (as specified in the exception workflow) are accepted (this is illustrated by Figure 68). Firstly because the exception workflow does not contain changes to the state of the business rules no new path of execution needs to be started nor should invalid paths of execution be removed from the standard workflow. But this particular workflow does specify changes to the output of a set of activities. Thus (A) the quantity of products ordered needs to be increased to 75 from 50, (B) in week 48 and week 49 respectively 15 and 10 stock items should be allocated to item 10.003.001.22, (C) the increased quantity of items should also be reflected in the order confirmation and (D) delivery 20 is set to the DAF65 truck instead of the SPRINTER van. Finally the lock on the data item QUANTITY from the output of activity A needs to be released, so any sales order processing activity that requires to use this data item can be started.



DEL. NO	TRUCK	DATE	ORDER
20	DAF65	15/12/2001	921718

WEEK	ITEM	ORDER	ALLO.
50	10.003.001.22	921718	30
49	10.003.001.22	921718	30
48	10.003.001.22	921718	15

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	75	14/12/2001	100	7500

ORDER	CUSTOMER	ITEM	DESCRIPTION	QUAN.	DATE	PRICE	AMOUNT
921718	23174	10.003.001.22	Dover Chairs	75	14/12/2001	100	7500

Figure 68: State of the standard workflow after the exception workflow has been applied.

7.0 Discussion

In section 3.1 of chapter 5 a set of end-user (or sales order processor requirements) were proposed that a workflow approach has to meet. These requirements were described with reference to the term 'completeness', namely:

- The proposed workflow approach should be capable of detecting those types of exceptions of relevance to sales order processing (i.e. logical activity failures, input changes, and business rule changes - as identified in chapter 5). The new workflow approach proposed meets this requirement.
- Various exceptional situations need to be handled adequately by the workflow approach. The proposed workflow approach can generate and recommend many appropriates sequence of compensation activities for the kinds of exceptions that may occur and the states a workflow may be in. It is capable of detecting invalid paths of execution (and activities for which results should be cancelled) and identifying new paths of execution needed. It is also capable of detecting those activities that must be redone and those activities for which execution should be temporarily halted.

  However some exceptional situations cannot be sufficiently well addressed, as follows:

  1. Logical failures of activities such as 'despatch order' and 'invoice customer' cannot be handled well by the new workflow approach. If they occur many planning activities will be triggered again for re-execution although in practice there may be no need for them to

be redone. Consider for example the case when goods are despatched from the warehouse to the customer but too few items are actually available. In this case another delivery may be made. Thus it is not necessary to check the creditworthiness of the customer because of a change to the sales order quantity nor is it necessary to re-allocate stock. This indicates a need to do further research into handling sales order processing exceptions after many planning activities have been performed.

2. When using the new workflow approach it can occur that an exception modifies the state of a path of execution so that it becomes invalid and this can cancel the results of many activities. But subsequently other or new paths of execution may require execution of these activities to be re-triggered resulting in additional work. Thus additional improvements could be made if it were possible to detect whether such activities may be triggered on other paths of execution. If not, they can be cancelled without any concern; otherwise additional business logic needs definition and application if and when activity results are cancelled.
3. The blocking mechanism is primitive and in practice its application may limit the progress of a workflow significantly. When an exception occurs some or all paths of execution are blocked for the duration of the exception; thus if many exceptions occur during the life-span of a workflow it may be that little progress is made. In fact the simple blocking mechanism deployed is very 'pessimistic', i.e. in case of any uncertainty no work may be performed if a chance exists that it may have to be redone. Indeed the technique does not take into account (A) the chance that is has to be redone and (B) the cost of the effort needed in case it should be redone. Thus it may be better to adopt a more 'optimistic' technique that lets a workflow progress, but blocks only those transitions of a workflow for which an exception is too costly to handle.

Thus much outstanding research can be undertaken to refine existing and introduce new rules with capability to generate appropriate compensation activities for given exceptions and workflow states.

- It must be possible to reverse the effects of an exception workflow. This requirement is also met by the new workflow approach. However a limitation of the new workflow approach is that exceptions cannot be evaluated in a combined way. In some cases another exception may occur before one is handled. Even though the effects of both exceptions can be evaluated independently from each other, it may interesting to combine the effects of two exceptions in a single exception workflow in order to determine whether both can be handled or whether the exceptions conflict. If evaluated independently from each other, sales order processors will have to do this manually. For example assume that exceptions A, B, C and D have occurred and that it may be interesting to evaluate exceptions A, B and C in a combined way, or to analyse the effect of exceptions B, C and D together. Thus there may be a need for multiple versions of a standard workflow to facilitate evaluation of different exception handling scenarios.

In section 3.2 and 3.3 of chapter 5 a set of workflow system developer requirements were proposed that a new workflow approach might advantageously meet. These requirements were considered with respect to criteria 'effort & flexibility' and 'correctness'. In the proposed approach appropriate compensation activities are automatically generated from knowledge about a control and a data flow specification, the state of a workflow and from properties of the particular exception. Thus it takes relatively little time for workflow system developers to develop 'exception handling specifications', i.e. data and control flow specifications. In the case of data flow specifications the use and produce of data was formulated in terms of data entities. In the case of the control flow specification it is observed that it is necessary to specify conditions about whether and when activities should be performed. In both cases the effort required was estimated as being linearly related to the number of activities involved in a workflow specification (i.e. the criterion effort). Consequently it is assumed that it will be relatively easy to modify such specifications in limited time (i.e. the criterion flexibility) when compared with previous workflow approaches studied in the literature. Finally it is also a relatively simple matter to check the new specifications for correctness as it is possible to focus on the flow of data or control.

On the other hand it may take a significant amount of time for workflow approach conceivers to improve upon existing workflow approaches and develop new ways of specifying workflow state and workflow rules needed to underpin exception handling. Thus the burden is shifted from workflow system developers to conceivers of the new workflow approaches.

8. A component-based workflow system

1.0 Introduction

A component-based workflow system was developed as part of this study to implement key aspects of the new workflow approach within the context of existing commercially available workflow and application system components. An important aim is to formalise the required interactions between system components so companies can select those components that suit their needs in the best way and system developers can integrate the components in simple way. At the end of this chapter some major implications for system developers will be discussed.

2.0 General architecture of the new workflow system

The developed workflow system comprises two interconnected systems, namely a 'workflow specification system' (illustrated by Figure 69) and a 'workflow enactment system' (illustrated by Figure 70). Both of these systems have been designed to incorporate state of the art implementation notions about software, components and architecture. The aim here has been to facilitate ongoing system development and application in specific instances of the sales order processing domain.

In its current state of development the 'workflow specification system' consists of a single component, namely a Workflow Specification Editor (WSE) component. This component defines standard workflow and exception handling specifications by means of (A) control flow specifications, (B) data flow specifications, and (C) state-transition specifications.

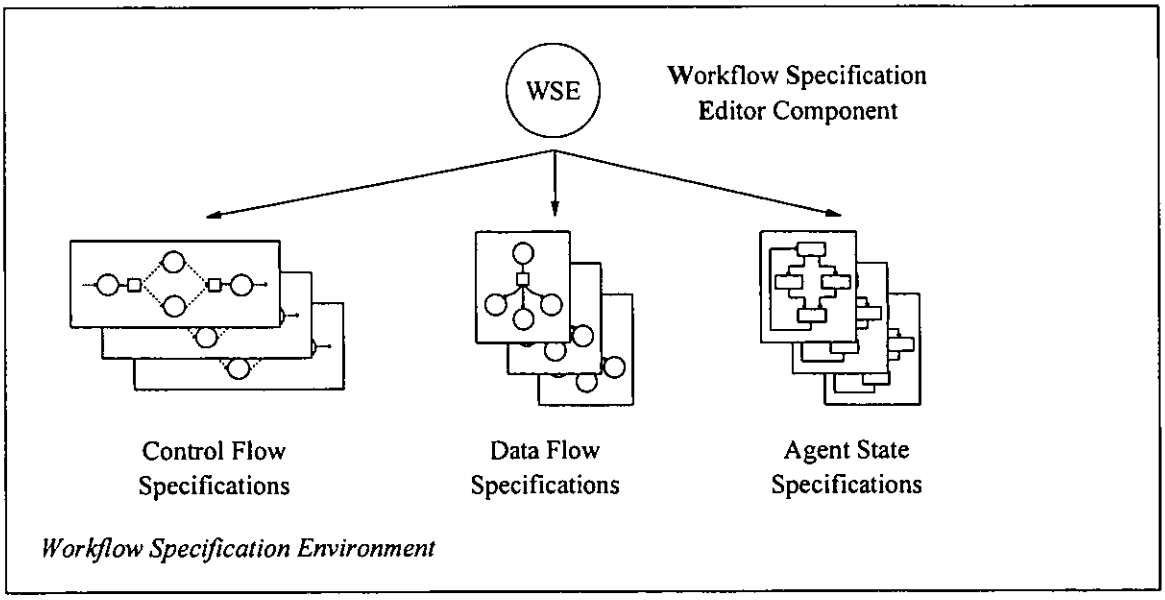

WSE
Workflow Specification Editor Component
Control Flow Specifications
Data Flow Specifications
Agent State Specifications
Workflow Specification Environment

Figure 69: A workflow specification system.

The current workflow enactment system comprises three kinds of components, namely (1) a Workflow Engine (WE) component, (2) a number of Agent (Ag) components and (3) Application (Ap) components associated with agent components. The purpose of the workflow engine component is to manage the execution of standard flows of work and to handle exceptions. Agent and application components function co-operatively to manage the output of individual workflow activities. Whereas all control and data flow specifications of a workflow system are stored and accessed by the workflow engine component, each agent component has a unique state-transition specification.

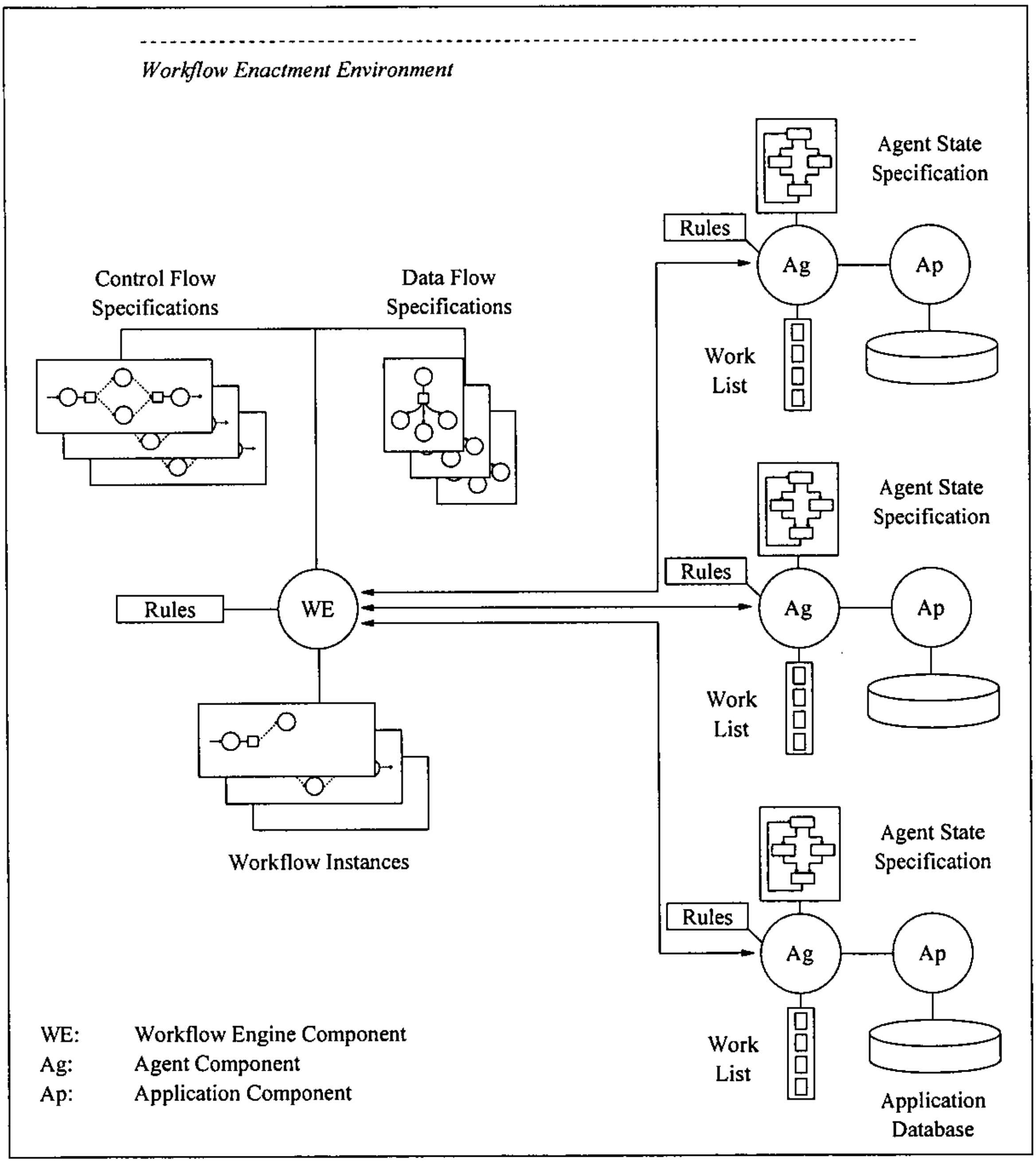

Workflow Enactment Environment
Control Flow Specifications
Data Flow Specifications
Rules
WE
Workflow Instances
Agent State Specification
Rules
Ag
Ap
Work List
Agent State Specification
Rules
Ag
Ap
Work List
Agent State Specification
Rules
Ag
Ap
Work List
Application Database
WE: Workflow Engine Component
Ag: Agent Component
Ap: Application Component

Figure 70: The workflow enactment system.

The workflow engine component and the various agent components of a configured workflow system have been designed so that they interact in the manner illustrated by Figure 71. An agent component informs the workflow engine component of the need to initiate a workflow. Such a class of event was classified as being an 'Enable External Precedence Link' event. To initiate workflow execution the workflow engine component instructs agent components to execute their standard activity in a particular order. Thereby 'Execute Standard Activity' events are signalled as input events to agent components. Each time an agent component completes a standard thread of activity (corresponding to what is termed a 'Standard Activity Executed' event) a number of other agent components may start executing their standard activity (each of which also corresponds to an 'Execute Standard Activity' event).

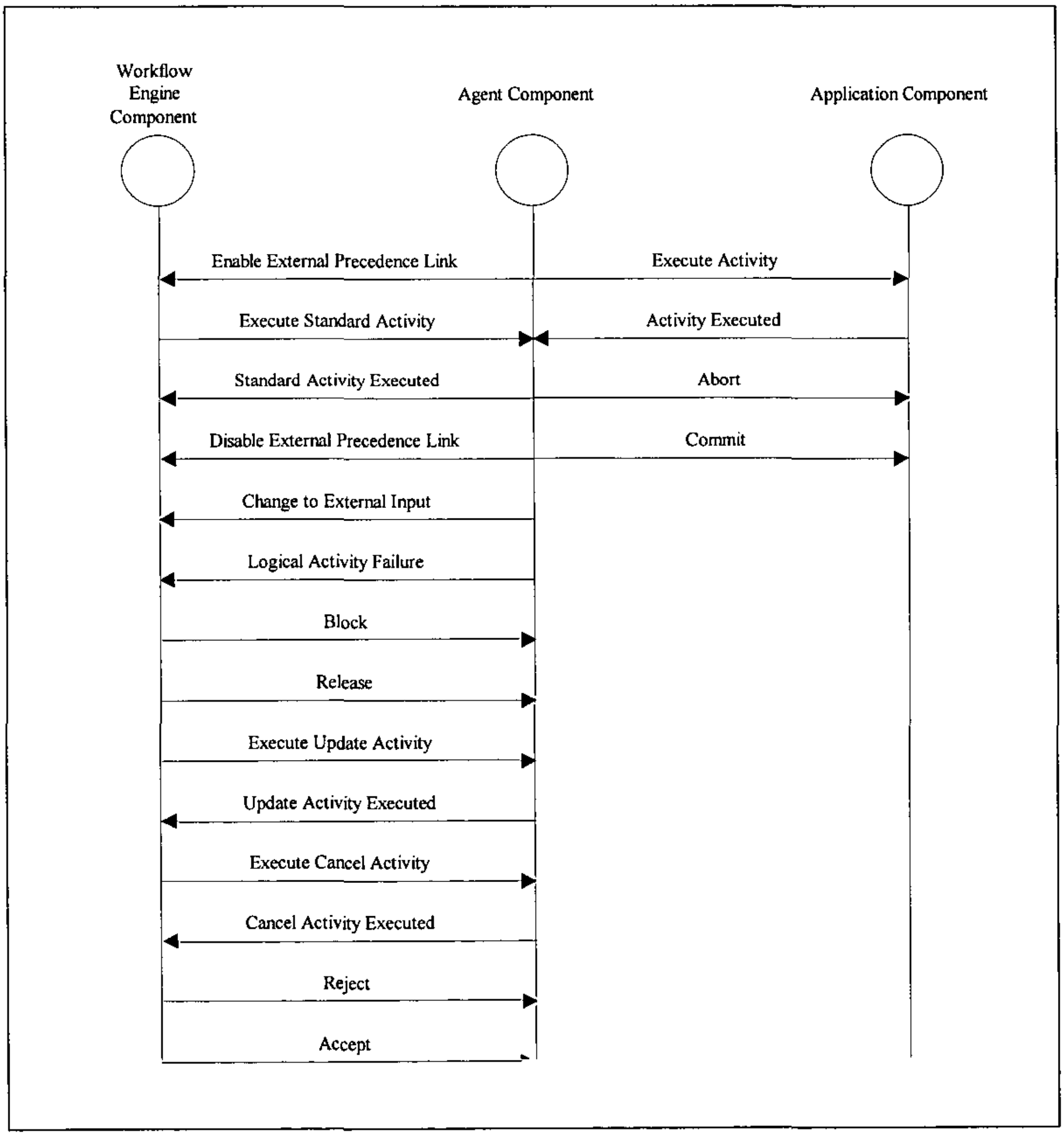


Figure 71: Exchange of events between system components.

However agent components may also inform the workflow engine component that one or more exceptions have occurred. An agent component may report a logical failure during the execution of a standard activity. Such an occurrence is termed a 'Logical Activity Failure' event. Alternatively an agent component may indicate to the workflow engine component that a change has occurred to the value of external input used by a standard activity. This alternative type of exception is termed a 'Change to External Input' event. Thirdly an agent component may request that a workflow should be cancelled, this being termed a 'Disable External Precedence Link' event. If either of the above three types of exception occurs a workflow engine component may prevent other agent components from temporarily performing their standard activities, i.e. it may generate 'Block' events. Later it may lift the blocking constraints imposed, i.e. by issuing a 'Release' event. Under conditions where an exception has occurred the workflow engine component determines which of those standard activities that have previously been performed need to be updated and cancelled. Based on results of its analysis it instructs agent components responsible for executing compensation activities, i.e. 'Execute Update Activity' and 'Execute Cancel Activity' events at those components. In due time agent components will report on the completion of compensation activities by generating 'Update Activity Executed' and 'Cancel Activity Executed' events. Further the workflow engine component commands agents will make the effect of some compensation activities permanent or will discard other compensation activities when all compensation activities related to specific exceptions have been processed. Here the workflow engine component issues an 'Accept' or 'Reject' event.

3.0 Developed workflow system components

3.1 Workflow engine component

The task of the developed workflow engine component is to manage the life-cycle of a set of workflows. The behaviour of this component is governed by (1) workflow specifications, (2) the occurrence and status of workflow instances and (3) a set of event types and rules.

3.1.1 Workflow specifications

The workflow engine component deploys a workflow specification structure which conforms to the logical workflow approach described in section 4 of chapter 7. Both data and control flow specifications used here adhere to definitions included into section 4 of chapter 7.

3.1.2 Workflow instance

Certain characteristic properties of workflow instances are managed by the workflow engine component in a manner that is largely consistent with logical workflow approach definitions discussed in section 5 of chapter 7. However an important difference is that within the workflow system the workflow engine is concerned primarily with synchronising activity execution and therefore it needs to register the state of activities, whilst application components need to possess capabilities to store the output of activities. Figure 72 shows how the state of a workflow instance is represented and stored by the workflow engine component without recording specific outputs of standard activities. Also in section 5 of chapter 7 it was argued that a workflow instance does not capture logical data flow dependencies (since these are already encoded in data flow specifications). Thus issues of encoding needed properties of any standard workflow are in effect reduced to encoding properties of control flow instances and the status of activities associated with control flow instances. Exception workflow representations utilised by the workflow engine correspond to definitions of exception workflow properties as defined by the logical workflow approach described in section 5.2 of chapter 7.

To aid the reader's understanding, a short review follows about relevant definitions included into section 5 of chapter 7. Here it was determined that a control flow instance should encode the state of the business rules (i.e. precedence links and junctions) that have been evaluated. Further the various states reached by a 'standard activity' can be defined as being *enabled*, *started*, *completed*, *logical failure* and *blocked* (where the semantics of these states are defined in Table 7). An exception workflow consists of a set of interrelated exceptions and solutions. A root exception is an exception that triggers an exception workflow, whilst a derived exception is an exception that has been directly triggered by a solution. A root exception can be instigated either by a logical activity failure or by a change to external information. From a technical perspective compensation actions generate solutions that consist of designated changes to the data used (also termed input) and/or data produced (also termed output) so that the new input will be consistent with the new output according to the logic of specific activities.

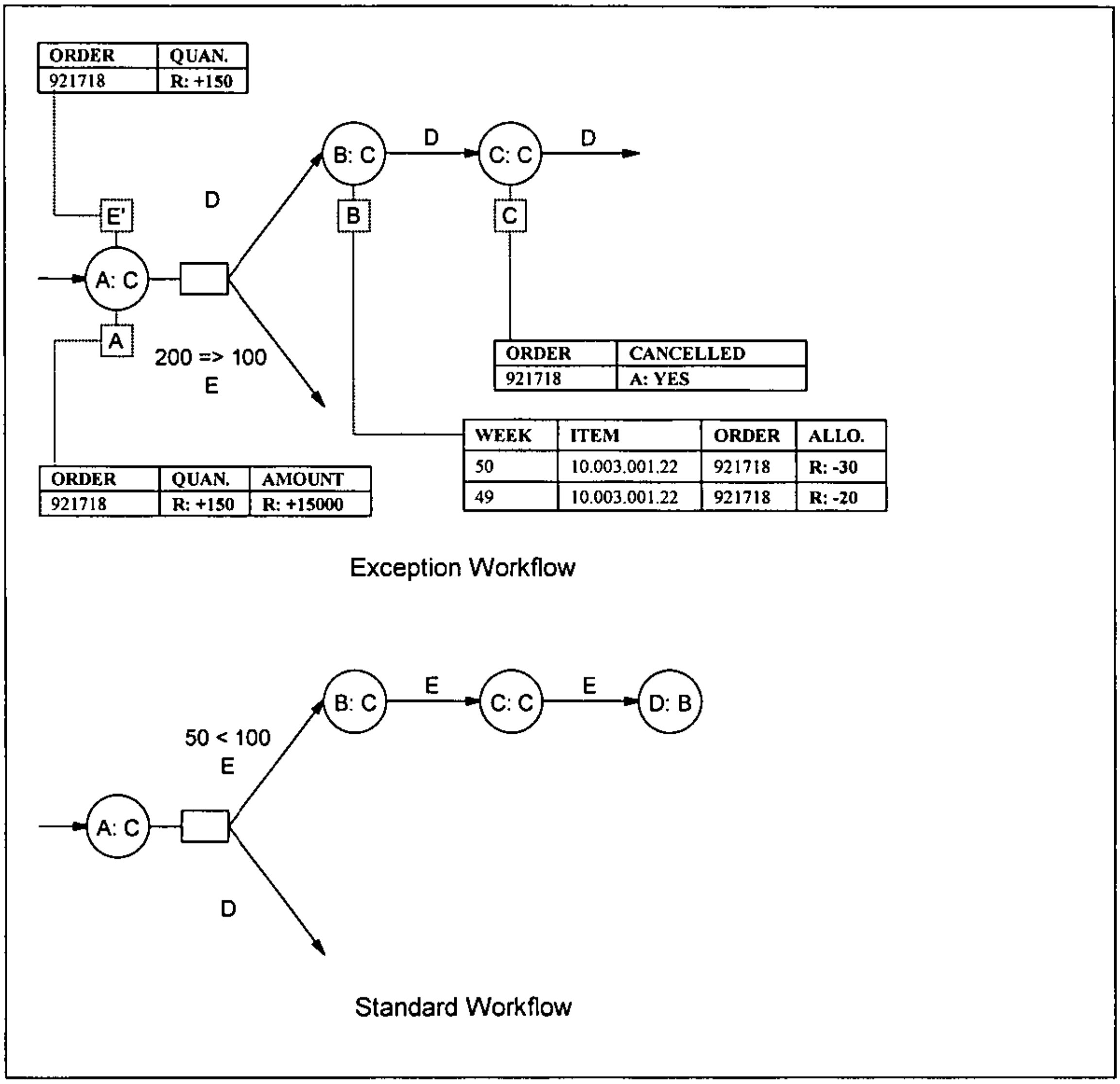


Figure 72: State of a workflow instance (as defined by its standard workflow and exception workflows).

Consider for example the exception workflow illustrated by Figure 72. This exception workflow has been triggered by a request from the customer to increase the ordered quantity for order 921718 by 150 (as defined by root exception E'). After compensation activity A ('Modify Sales Order Properties') has processed this exception, its output is modified (as defined by solution A). As a consequence two derived exceptions are propagated to activities 'Allocate Stock' and 'Send Order Confirmation'. As a consequence the output of these two activities requires adjustment. Firstly any items allocated to order 921718 will need to be de-allocated. Secondly an order cancellation needs to be sent.

3.1.3 Workflow transformations

In this subsection the actions performed by the workflow engine component are explained in terms of data and control flow specifications, changes to the state of workflow instances and events that are generated.

3.1.3.1 Management of standard workflow instances

The workflow engine component updates a standard workflow instance in response to two events: 'Precedence Link Enabled' and 'Standard Activity Executed'. It sets the state of activities, precedence links and junctions and generates events for agent components.

1. 'Precedence Link Enabled'. When a precedence link becomes enabled the workflow engine component will respond by (1) creating a new standard workflow instance and (2) determining which enabled activity should be executed first. With respect to the state of the standard workflow instance, the state of the precedence link and the activity state is set to 'Enabled'. Also at the agent component that manages this activity type an 'Execute Standard Activity' event is generated.
2. 'Standard Activity Executed'. When a standard activity reaches completion, the workflow engine component will set the state of the activity to the state 'Completed'. Following which the workflow engine component will determine whether other precedence links and fan-in junctions have become enabled and will generate 'Execute Standard Activity' events for those activities that have been enabled for execution. The workflow engine component refers to the control flow specification to determine those activities that are enabled for execution and refers to the standard workflow instance to decide which activities have yet to be started.
   If all activities have been completed, then execution of the standard workflow is complete, so that the standard workflow can be shelved.

3.1.3.2 Management of exception workflows

The workflow engine component updates an exception workflow instance in response to seven events: 'Disable External Precedence Link', 'Logical Failure', 'External Data Change', 'Update Activity Executed', 'Cancel Activity Executed', 'Accept' and 'Reject':

1. 'Disable External Precedence Link'. When a precedence link becomes disabled the workflow engine component creates a new exception workflow and cancels execution of the standard workflow. As the precedence link that triggered the standard workflow is no longer enabled all existing paths of execution become invalid. In the case of the exception workflow the state of all precedence links and fan-in junctions will be set to 'Disabled' and the state of previously enabled activities will be set to 'Cancel: Enabled''. Agent components that are responsible for these activities will require an 'Execute Cancel Activity' event.

2. 'Logical Failure' and 'External Data Change'. Upon receiving notification of a root exception (of either a 'Logical Failure' or 'External Data Change' type) the workflow engine component: (1) creates a new exception workflow, (2) stores the status of the root exception for the activity concerned and (3) sends an 'Execute Update Activity' trigger signal to the agent component.
3. 'Update Activity Executed' and 'Cancel Activity Executed'. For either of these types of event the workflow engine component will determine (A) those activities in the standard workflow that should be blocked temporarily, (B) those existing paths of execution that have become invalid and those new paths of execution that may be started, and (C) which up- or downstream activities should be updated. For activities that (A) are to be blocked temporarily, then 'Block' events are created, (B) lie on invalid paths of execution, then 'Execute Cancel Activities' are generated and (C) are to be updated, then 'Execute Update Activities' are produced.
4. 'Accept'. In cases of all exceptions being accepted an internal event is generated which indicates that an exception has been accepted. Under such conditions, the workflow engine component will (A) apply all needed changes (encoded by the exception workflow) to the standard workflow, (B) generate 'Accept' events for all updated and cancelled activities in the exception workflow, (C) create 'Execute Standard Activity' events for those activities that have been enabled for execution and (D) shelve the exception workflow. In cases of point (A) the workflow engine component copies all state changes to precedence links, fan-in junctions and activities from the exception workflow to the standard workflow, thereby updating the standard workflow. With respect to point (B) 'Accept' events will trigger the agent component and thereby will make permanent needed changes to activity output generated by compensation activities. For activities that have first been enabled (point C) agent components will first need to create an activity.
5. 'Reject'. In cases of an exception being rejected an internal event is generated within the workflow engine. In such cases the workflow engine component will (A) generate 'Reject' events for all updated and cancelled activities encoded by the exception workflow, (B) generate 'Release' events at activities for which execution has been blocked and (C) shelve the exception workflow. With respect to point (B) 'Release' events will trigger the agent component so that it discards changes to activity output generated as a consequence of compensation activities. For activities that have been blocked (point C) agent components can again perform their role.

3.2 Agent component

The task of agent components is to manage the state of activities associated with each activity type. For example one kind of agent component may be responsible for the state of activities of the activity type 'Maintain Sales Order Properties' or another for the activity type 'Send Customer Order Confirmation'. However a single agent component cannot be responsible for both of these activity types. Also it has Instances of each agent component may be invoked many times as typically it will have to perform its role for many sales orders such as sales order SO153, SO158, SO202, etc. Hence in this study agent components were specified and developed with capabilities to manage the states that an activity may reach and the actions that should be performed when each state is reached. Without proper guidance about managing the state of an activity any given end user may forget to perform all necessary actions or may initiate execution of wrong actions. In general for a Assemble-To-Order (ATO) company the number of orders will be large and thus human sales order processors may experience difficulties in remembering all outstanding actions for different orders. Within the proposed architectural framework a key function of the agent component is to remind end users about the standard and compensation activities that remain to be performed for orders in their current state. Whereas an agent can ensure, at a single logical unit of activity, that performance of standard and compensation activities is carried out in a defined order, a workflow engine component treats activities as basic, indivisible elements in a flow of work. Therefore unlike a workflow engine component, an agent component is not concerned about specifics of the role it plays in a particular flow of work.

An agent component may be viewed conceptually as a work list handler. This functionality is akin to that proposed by the WfMC [WFM94]. However the functionality of a WfMC work list handler is rather limited when compared to the functionality of the developed agent components that operate within the proposed architecture. Whereas a WfMC work list handler removes a work item from its list after it has executed an activity, an agent component maintains a record of a work item until a given workflow completes, because subsequent exceptions may occur that require reasoning about and enactment of compensation activities. Only when a workflow completes, and therefore no further exceptions can occur, are all agent components purged in terms of activities and work items (in WfMC jargon) related to this workflow. Therefore the developed agent components have broader responsibilities than work list handlers defined by WfMC.

3.2.1 State

It was decided that the state of activities at agent components could be distinguished in terms of the progress these agent components have made as they perform standard, update and cancel activities. In only a few cases can we expect the execution of any activity to be immediate and its outcomes are successful. Usually a request to perform an activity is not immediately handled. Nor will an activity always complete successfully. Hence it is important to define appropriate sets of states that can code up the progress of execution of each activity so that appropriate remaining actions can be carried out in an ordered way. Such a set of states was determined and adopted as defined by Table 10.

Activity Type	Defined states of progress made by agent components
STANDARD	ENABLED; STARTED; COMPLETED; BLOCKED; FAILED
UPDATE	ENABLED; STARTED; COMPLETED
CANCEL	ENABLED; STARTED; COMPLETED

Table 10: Activity types and states.

The codes states of activities serves to remind an end-user (e.g. human sales order processor) that certain activities still need to be performed (e.g. are ENABLED, STARTED, or have a related LOGICAL FAILURE) or may temporarily not have been performed (i.e. BLOCKED). Whereas the state COMPLETED does not serve to remind a clerk but is required for the purpose of establishing a complete and consistent state definition. A semi-automated sales order system might use this information (about these generic states) to constrain the actions of clerks.

By adopting such an approach developed agents can perform these three kinds of activity for a workflow, namely standard, update or cancel activities. However, there will be constraints imposed on the order in which these three types of activity have to be performed. Under some conditions a standard activity can be performed, work goes according to plan and input data sets are properly converted to output data sets. When an update or a cancel activity is performed this must be (1) after a standard activity has been performed, and (2) in response to the occurrence of an exception. Figure 73 illustrates allowable logical sequences in which these activities can be executed.

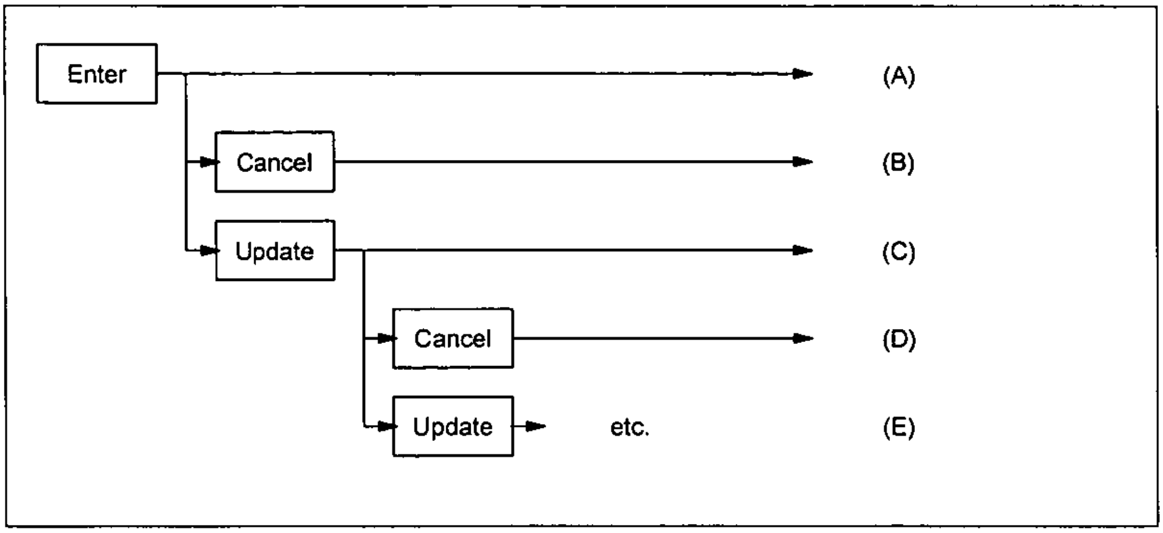


Figure 73: Logical sequences of standard and compensation activities.

No compensation activity can be performed before a standard activity has been performed. This is because compensation activities act upon inputs and outputs generated by standard activities. A standard activity does not necessarily have to be followed by a compensation activity, because exceptions do not always occur during the life span of a workflow (as illustrated by Figure 73.A). An update compensation activity may be followed by other (update and cancel) compensation activities, as illustrated by Figures 73.C, D and E. However after a cancel activity has been performed no compensation activity may follow, as illustrated by Figure 73.B.

It is also important to point out that the generic states of activities will need to have a specific meaning to clerks responsible for performing them. Examples of specific meanings of states for the activity 'Allocate Stock' are given in Table 11.

Generic state	**Specific state**
1.A Standard; Enabled	Need to **allocate** stock items to sales order
1.B Standard; Started	**Allocating** stock items to sales order
1.C Standard; Completed	**Sufficient** stock items **allocated** to sales order
1.D Standard; Logical Failure	**Insufficient** stock items **allocated** to sales order
1.E Standard; Blocked	**Wait** to allocate stock items to sales order
2.A Update; Enabled	Need to **re-allocate** stock items
2.B Update; Started	**Re-allocating** stock items
2.C Update; Completed	Stock items **re-allocated**

3.A Cancel; Enabled	Need to **de-allocate** stock items
3.B Cancel; Started	**De-allocating** stock items
3.C Cancel; Completed	Stock items **de-allocated**

Table 11: Generic and specific state definitions.

An activity does not necessarily have to be processed immediately. For example it may be that a number of instances of an activity will be performed together at periodic intervals. Alternatively the activity may be performed when a particular number of instances has been reached. In other situations it may be that instances compete with each other (i.e. for resources) in order to be processed and therefore may be assigned a priority. Concrete examples are (1) sending order confirmations to customers every day, (2) checking the creditworthiness of a customer in batches of ten orders, and (3) allocating stock to orders on the basis of the highest priority.

3.2.2 Transformation

The execution of activities at the developed agent components is governed by (A) state-transition specifications, (B) the current state of activities and (C) the occurrence of certain types of event. A state-transition specification defines the states that an activity may reach and the actions that must be taken at each state in response to events. Valid changes in state are specified by means of transitions. Transitions define (a) the kind of events that may occur when certain states are reached, (b) the actions that should be taken in response to these event types and (c) new states that can be reached next. Typical actions that can be initiated by an agent component will include: execute a standard or a compensation activity, and reverse the effect of a compensation activity or make the effect permanent. Thus the developed agent components handle events generated by users, its application component and the workflow engine component by:

1. retrieving the state of an activity at the instant when the event occurred and referencing the state-transition specification to determine the set of applicable transitions for this state. If the event is not specified by one of the transitions then it is discarded.
2. executing the actions (if any) associated with the event. This usually results in an event of interest to the workflow engine component or the application component.
3. updating the state of an activity to a new state determined by associated transitions.

Consider for example the state-transition specification illustrated by Figure 74. This specification defines nine states that a logical activity may reach and thirteen possible transitions. Assume that when the exception 'Execute Update Activity' occurs the state is 'Standard: Complete'. Associated with this state are two transitions (1. E: Execute Update Activity; A: ; and 2. E: Execute Cancel Activity; A: ;). Indeed the event may occur for the current state of the activity. In this case there is no action to execute. The new state of the logical activity becomes 'Update: Enabled'. Thus agent components process events in a universal way in as much that they do not have specific rules and mechanisms linked to each event. This is contrary to the case of the workflow engine component.

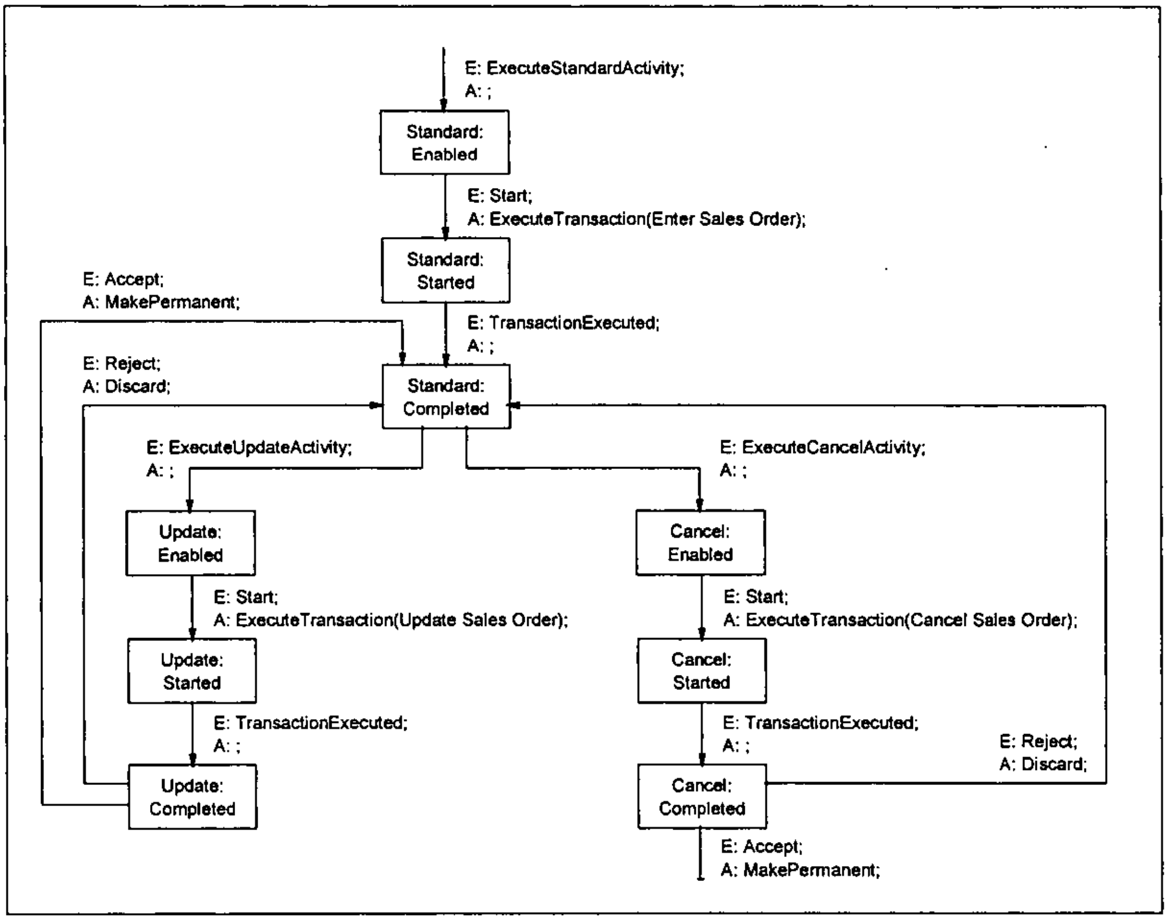


Figure 74: Typical state-transition specification.

A generic state-transition specification (such as the one shown in Figure 74) can assist workflow modellers when they define a context-dependent state-transition specification for each logical activity comprising a workflow. In general not necessarily all of these theoretical states nor transitions may be of relevance to a particular activity. For example it may not be necessary to model the state (Standard; Logical Failure) if the execution of an activity cannot fail in such a way. Also the state (Standard; Enabled) does not need to be modelled if the execution of an activity is to be started immediately. Naturally as the number of possible states is reduced so too is the number of possible state transitions. Thus the workflow modeller can evaluate the applicability of each generic state with respect to each activity, i.e. determine if states have meaning within the operating context of a particular logical activity or whether they can be discarded from the specification. Where a generic state is not found to be applicable and is not incorporated into a specific state transition specification, then the transitions that leave and enter this state do not apply and therefore they should not be incorporated into the specification.

3.3 Application component

The role of application components designed and developed in this study is to manage the output of an activity. An application component generates and manipulates the output of an activity by means of standard, update and cancel activities and stores the output in its database. A standard activity generates for a given set of inputs a new set of outputs. Following a given set of changes to the input of a previously performed standard activity appropriate update and cancel compensation activities are generated that enable the required set of changes to the output to be made so that the new input corresponds to the new output[41]. The changes generated by a compensation activity (with respect to the original input or output of a previously performed standard activity) are not immediately incorporated into the input or output by the application component. Rather these changes are kept separate for as long as workflow uncertainty exists with respect to any outstanding need to handle aspects of exceptions. If it becomes clear that there is no such need to take compensating actions the application component will simply discard the changes, otherwise the application component will apply the changes to the original version and thereby generate new versions of all relevant activity outputs.

The behaviour of the developed application components is simple and is not governed by any specification. As illustrated by Figure 71, an application component can simply be instructed by its agent component to do one of the following things: (1) execute a standard, update or cancel activity (event Execute Transaction), (2) make the changes to an output permanent (event Commit) or (3) discard changes invoked by compensation activity (event Abort). On transaction completion, an application component will report this event to its agent component, i.e. by issuing a Transaction Executed event.

Within the proposed workflow system architecture standard, update and cancel activities are implemented by means of transactions. Transactions consists of sequences of low-level operations specified in some programming language (such as for example C/C++). A transaction can read the records generated by another activity (or it can use data provided by the real system) and store its output records in the database, or it can flush data into the real system.

---

[41] Also update compensation activities can generate required changes to the input to enable correspondence with a given set of changes to the output. This feature is opposite to traditional thinking that an activity generates an output that is dependent on the input.

4.0 Example exchange of events between workflow system components

This section illustrates examples of event exchange amongst workflow engine, agent and application components as they function co-operatively to handle an exception.

4.1 Component specifications

Whilst the behaviour of a workflow engine component is governed by control and data flow specifications, the behaviour of agent components is governed by state-transition specifications. The control flow and data flow specifications used by the workflow engine component to control the execution of a workflow, have previously been defined by Figures 58 and 59. The agent component specifications are described under 8.3.1.1, 8.3.1.2 and 8.3.1.3 as follows.

4.1.1 Agent component 'Maintain Sales Order Properties'

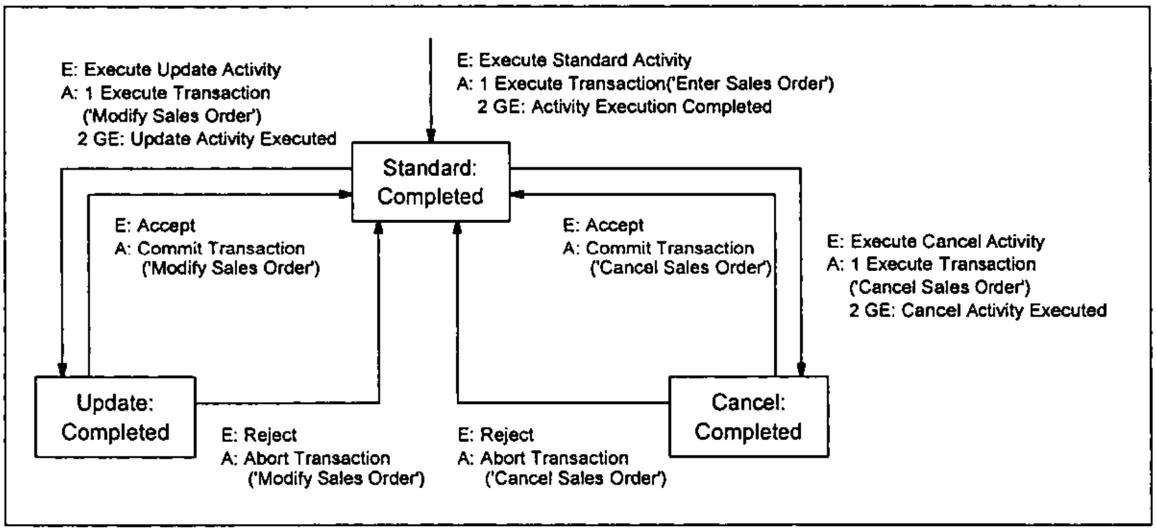

E: Execute Update Activity
A: 1 Execute Transaction ('Modify Sales Order')
2 GE: Update Activity Executed
E: Execute Standard Activity
A: 1 Execute Transaction('Enter Sales Order')
2 GE: Activity Execution Completed
Standard: Completed
E: Accept
A: Commit Transaction ('Modify Sales Order')
E: Accept
A: Commit Transaction ('Cancel Sales Order')
E: Execute Cancel Activity
A: 1 Execute Transaction ('Cancel Sales Order')
2 GE: Cancel Activity Executed
Update: Completed
E: Reject
A: Abort Transaction ('Modify Sales Order')
E: Reject
A: Abort Transaction ('Cancel Sales Order')
Cancel: Completed

Figure 75: Specified states of the agent component 'Maintain Sales Order Properties'.

1. Activity state 'Standard: Enabled' is not included in the state-transition specification as a sales order is entered immediately by the user. Further the state 'Standard: Started' is not included since (A) the execution of this activity cannot fail logically and (B) there is no need temporarily halt execution of this activity. Thus neither of the activity states 'Standard: Suspended' and 'Standard: Failed' are included. Indeed if the standard activity is executed, the transaction 'Enter Sales Order' is executed and the workflow engine component is informed of the new activity state (i.e. 'Completed').
2. In cases where activity results are to be updated or cancelled a compensation activity needs to be executed to record the new properties of a sales order (compensation activity 'Re-Allocate Stock') or to set a flag that the sales order needs to be cancelled (compensation activity 'De-Allocate Stock'). If an exception is accepted by all agent components the results are made

permanent via a commit transaction. However if an exception is rejected the changes are discarded via an abort transaction.

4.1.2 Agent component 'Allocate Stock'

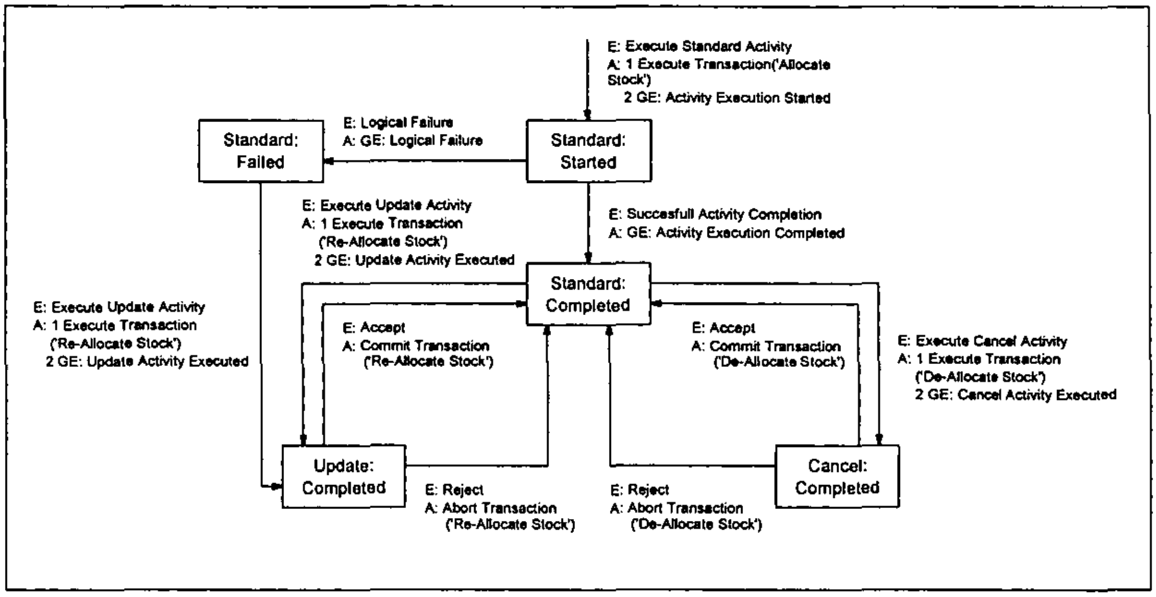

E: Execute Standard Activity
A: 1 Execute Transaction('Allocate Stock')
2 GE: Activity Execution Started
Standard: Failed
E: Logical Failure
A: GE: Logical Failure
Standard: Started
E: Execute Update Activity
A: 1 Execute Transaction ('Re-Allocate Stock')
2 GE: Update Activity Executed
E: Succesfull Activity Completion
A: GE: Activity Execution Completed
E: Execute Update Activity
A: 1 Execute Transaction ('Re-Allocate Stock')
2 GE: Update Activity Executed
Standard: Completed
E: Accept
A: Commit Transaction ('Re-Allocate Stock')
E: Accept
A: Commit Transaction ('De-Allocate Stock')
E: Execute Cancel Activity
A: 1 Execute Transaction ('De-Allocate Stock')
2 GE: Cancel Activity Executed
Update: Completed
E: Reject
A: Abort Transaction ('Re-Allocate Stock')
E: Reject
A: Abort Transaction ('De-Allocate Stock')
Cancel: Completed

Figure 76: Specified states of the agent component 'Allocate Stock'.

1. The state 'Standard: Enabled' has not been included because this kind of activity can be executed automatically without attention of the user. The state 'Standard: Failed' has been included since it may not be possible to allocate sufficient stock to a sales order and therefore the activity may fail logically. To encode the reaching of either the 'Standard: Failed' or 'Standard: Completed' it was found to be necessary to incorporate a specification of the state 'Standard: Started' state even though there is no need to suspend the execution of the 'Allocate Stock' activity. The state 'Standard: Suspended' has not been included as it does not make sense to temporarily interrupt the execution of this automated activity. Typically the duration of this automated activity is likely to be in the order of only hundreds of a second.

2. In cases where the results of the 'Allocate Stock' activity are to be updated or cancelled a compensation activity needs to be executed. Thus it is necessary to determine the impact of such an exception. If it is determined that the results are to be updated then the compensation activity 'Re-Allocate Stock' is executed. Whereas the compensation activity 'De-Allocate Stock' is executed if the results are to be cancelled. If an exception (and its determined compensating actions) is accepted by all agent components the results are made permanent via a commit transaction But if an exception is rejected the changes are discarded via an abort transaction. Finally if the execution of an allocate stock activity fails an update activity may be started.

4.1.3 Agent component 'Confirm Order'

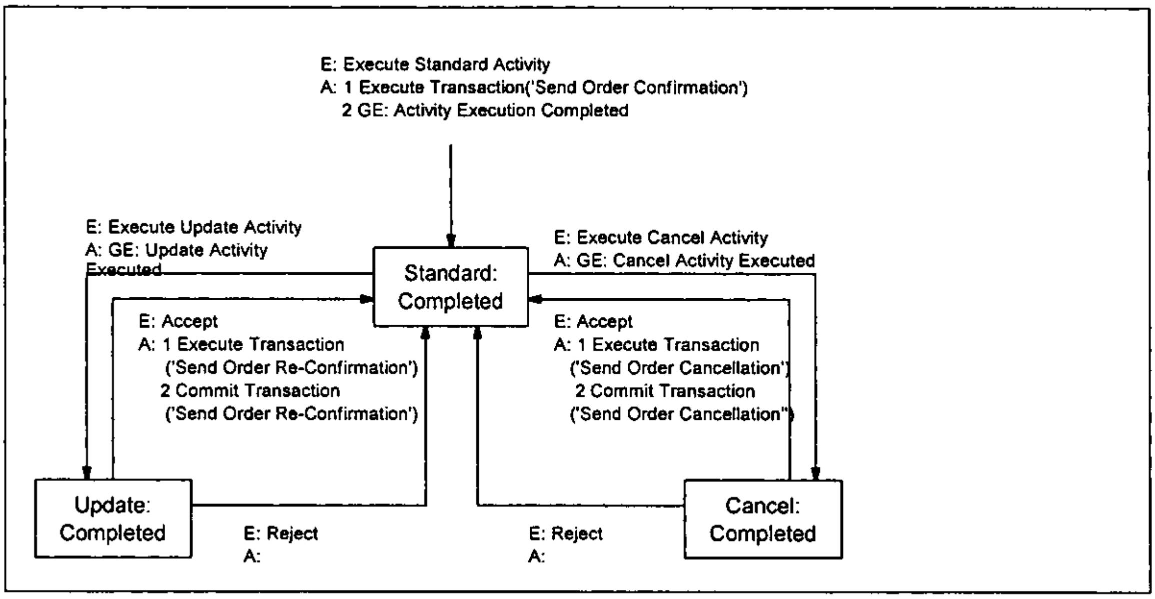

E: Execute Standard Activity
A: 1 Execute Transaction('Send Order Confirmation')
2 GE: Activity Execution Completed
E: Execute Update Activity
A: GE: Update Activity Executed
Standard: Completed
E: Execute Cancel Activity
A: GE: Cancel Activity Executed
E: Accept
A: 1 Execute Transaction ('Send Order Re-Confirmation')
2 Commit Transaction ('Send Order Re-Confirmation')
E: Accept
A: 1 Execute Transaction ('Send Order Cancellation')
2 Commit Transaction ('Send Order Cancellation')
Update: Completed
E: Reject
A:
E: Reject
A:
Cancel: Completed

Figure 77: States for agent component 'Confirm Order'.

1. Activity state 'Standard: Enabled' is not included into the state-transition specification of the 'confirm order' agent component as an order confirmation can be send automatically (and does not need to be triggered by the user). Also the state 'Standard: Started' is not included since (A) execution of this kind of activity cannot fail logically and (B) there is no need to temporarily halt execution of this activity, thus there is no need to specify the 'Standard: Suspended' state.

2. In cases where the results of the 'confirm order' activity are to be updated or cancelled there is no need to take compensating action when an exception occurs. As, first of all, the total impact of an exception on all agent components needs to be determined, the agent component may not send an order cancellation nor an order re-confirmation to the customer. This may only be done if an exception has been accepted by the workflow engine. Therefore after the transaction is executed a commit transaction needs to follow immediately. In cases where an exception is rejected the state of the activity is simply returned to its original state (i.e. 'Standard: Completed') without any further action being taken.

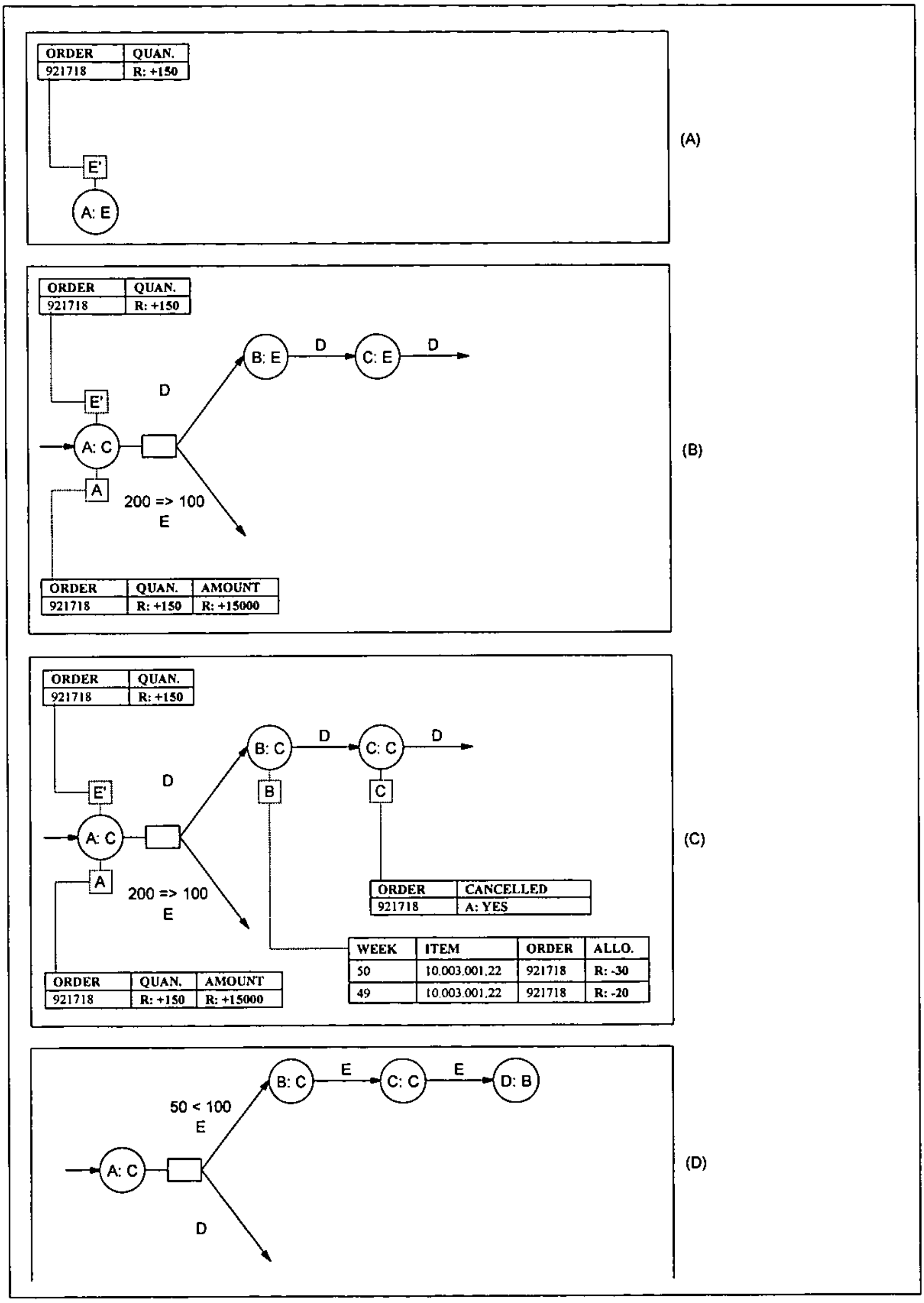
4.2 Component interaction

Figure 79 illustrates over time the event interaction between developed workflow system components when an exception is handled. Interaction is required between a workflow engine (WE) component, three agent (Ag) components and their application (Ap) components. The workflow engine component interacts only with agent components A, B and C. While agent components A, B and C communicate with their application components A, B and C respectively. Events generated by one component can become inputs to another component. This phenomenon is indicated by vertical arrows starting at one component and completing at others. Table 12 gives the properties of all event types that can occur.

The handling or processing of events by components is indicated graphically via squares and a designated number:

1. An 'External Input Change' event is generated by the agent component A to inform the workflow engine component that external information (used as an input to an activity) has been changed.
2. In response to an event described under 1 above, the workflow engine component (A) creates a new exception workflow, (B) stores the properties of the root exception and sets the state of the compensation activity to 'Enabled' and (C) generates an 'Execute Update Activity' event for agent component A. The state of the exception workflow in such cases is illustrated by Figure 78.A.
3. Agent component A requests its application component to execute the compensation activity 'Modify Sales Order' and informs the workflow engine component to update the state of the compensation activity to the 'Completed' state in the exception workflow.
4. Application component A executes the transaction 'Modify Sales Order'.
5. The workflow engine component (A) sets the state of the compensation activity A to 'Completed', (B) determines from the evaluated solution that the upper path of execution has become invalid whilst the lower path of execution has become valid, (C) sets the state of the precedence links that have become invalid (i.e. to Disabled) and (D) generates 'Execute Cancel Activity' events at agent components B and C and a 'Block' event for agent component D (which is not shown in this figure). The state of the exception workflow at this stage of exception handling is shown by Figure 78.B.
6. Agent component B requests its application component to execute the compensation activity 'De-Allocate Stock' and informs the workflow engine component to update the state of the compensation activity (in the exception workflow) to the state 'Completed'.
7. Application component B executes the transaction 'De-Allocate Stock'.
8. Agent component C requests its application component to execute the compensation activity 'Send Order Cancellation' and informs the workflow engine component that it should update the state of the compensation activity in the exception workflow to 'Completed'.
9. Application component C executes the transaction 'Send Order Cancellation'.

10. The workflow engine component (A) sets the state of the compensation activity B to 'Completed', (B) stores the solution of compensation activity and (C) determines that no derived exception has resulted but that not all compensation activities have been completed.
11. The workflow engine component (A) sets the state of the compensation activity C to 'Completed' and stores the evaluated solution of compensation activity, (B) determines that no derived exception has resulted and that all compensation activities have been completed, and (C) sends commit events to agent components B and C. At this stage of exception handling the state of the exception workflow will be as shown by Figure 78.C.
12. The agent component initiates a commit state for the executed transaction 'De-Allocate Stock'.
13. The agent component initiates a commit state for the executed transaction 'Send Order Cancellation'.
14. Application component B makes permanent the results of the transaction 'De-Allocate Stock'.
15. Application component C makes permanent the results of the transaction 'Send Order Cancellation'.



ORDER	QUAN.
921718	R: +150

ORDER	QUAN.
921718	R: +150

ORDER	QUAN.	AMOUNT
921718	R: +150	R: +15000

ORDER	QUAN.
921718	R: +150

ORDER	CANCELLED
921718	A: YES

WEEK	ITEM	ORDER	ALLO.
50	10.003.001.22	921718	R: -30
49	10.003.001.22	921718	R: -20

ORDER	QUAN.	AMOUNT
921718	R: +150	R: +15000

Figure 78: Three states of exception workflows A, B and C and standard workflow D.

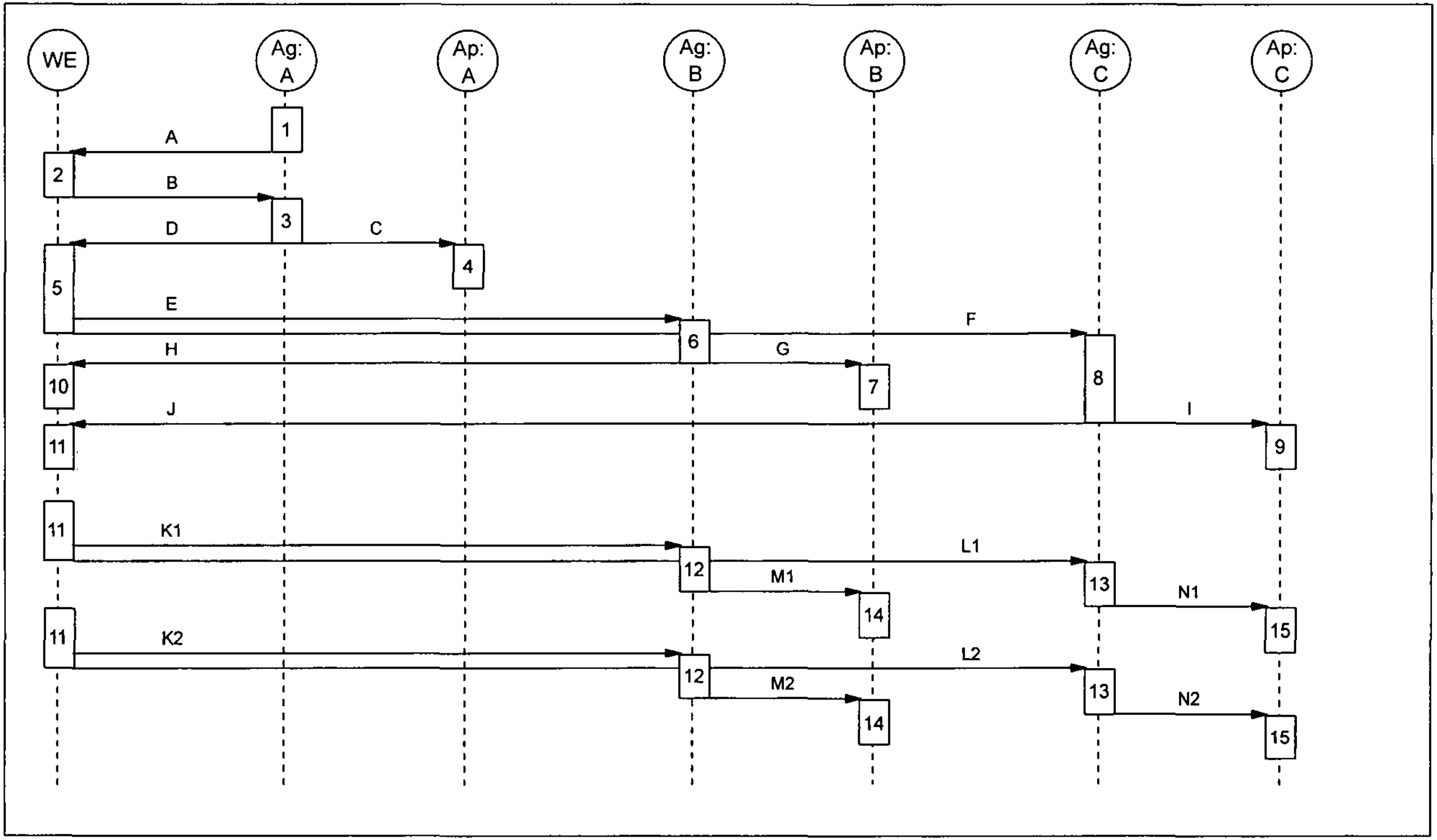


Figure 79: Event interaction between workflow components.

A	External Information Change (“Enter Sales Order”, “SO 100”, “Quantity: -50”)	
B	Execute Update Activity (“Modify Sales Order”, “SO 100”, “Quantity: -50”)	
C	Execute Transaction (“Modify Sales Order”, “SO 100”, “Quantity: -50”)	
D	Update Activity Executed (“Modify Sales Order”, “SO 100”)	
E	Execute Cancel Activity (“De-Allocate Stock”, “SO 100”)	
F	Execute Cancel Activity (“Send Order Cancellation”, “SO 100”)	
G	Execute Transaction (“De-Allocate Stock”, “SO 100”)	
H	Cancel Activity Executed (“Modify Sales Order”, “SO 100”)	
I	Execute Transaction (“Send Order Cancellation”, “SO 100”)	
J	Cancel Activity Executed (“Send Order Cancellation”, “SO 100”)	
K	1. Accept (“SO 100”)	2. Commit (“SO 100”)
L	1. Accept (“SO 100”)	2. Commit (“SO 100”)
M	1. Reject (“SO 100”)	2. Abort (“SO 100”)
N	1. Reject (“SO 100”)	2. Abort (“SO 100”)

Table 12: Events and their properties.

5.0 Discussion

This research activity has mapped the logical workflow approach (as described in chapter 7) onto standard WfMC components and extended the WfMC 'component interaction' specification in order to characterise possible implications for workflow component builders. Firstly workflow structures and rules were partitioned with respect to workflow engine, work list handler (or agent) and application workflow components[42]. Secondly events were defined that components may generate for other components in order to convey information about exception handling. As a consequence two new specifications resulted that may be of relevance to the development of next generation component-based workflow systems, namely:

1. A component-interaction specification (as discussed in section 2.0 and defined by Figure 71).
2. Example component implementation specifications (as discussed in sections 3.1, 3.2 and 3.3).

An important achievement of the component-interaction specification is that actions on a sales order by a workflow engine and agent components are synchronised and tied to the occurrence of (exception) events. To date the reference model of the Workflow Management Coalition defines a basic level of technical integration amongst workflow engine, work list handler and application components in order to let components access services from other components [WFM94]. This definition defines common services such as 'get work list' and 'get work item'. However little work has been done by the WfMC to define when and how these services should be invoked.

Even though component implementations are a matter for the vendor (who may decide on the programming language, programming structures and services, etc.) their implementation may be greatly facilitated by examples of data structures, functions and explanations. It follows that this research has identified and part satisfied the need for important specification extensions for existing software components, and namely the following components:

1. Workflow engine component. Two important extensions will be required. Firstly workflow instance structures will need to be extended to include exception workflows. Secondly many workflow system mechanisms will need to be developed to manage exception workflows. In particular the second extension is likely to require a considerable development effort. Whilst the provision of execution logic to manage standard workflows should be relatively simple, in addition complex mechanisms need developing that take account of (A) the actual state of a

---

[42] It is not an aim of the Workflow Management Coalition to standardise the functionality of workflow components. However before common ways of component interaction can be defined it is necessary to understand how a logical workflow approach maps onto workflow system components.

workflow, (B) properties of the exception and (C) data and control flow dependencies between workflow activities.

2. Application component vendors will need to make relatively minor software modifications to enable their components to fit in the proposed framework. Application components will need to be extended to capture, store and apply different activity output versions at any point in time. The current generation of applications record new values associated with activity outputs so that they can replace old values when a transaction commits. However in the new workflow approach many activity output versions may exist over time and must be recorded as relative operations on data, in order that these changes can be applied in any order. However it must be accepted that the current generation of applications already have good capabilities to link changes to different records via the use of transactions.
3. Workflow list handlers. Work list handlers will need to be extended to (A) prevent changes to workflow activity states under some conditions and (B) to pass on commands from workflow engine to application components and vice versa.

Finally some work still needs to be done to complete this research activity. The component-interaction specification needs to integrated with that of the workflow specifications already developed by the WfMC.

9. Idealised sales order processing domain study

1.0 Introduction

This chapter will (A) outline important steps necessary to quantity the extent of the capabilities and limitations of existing workflow approaches and the new approach for sales order workflows and (B) illustrate these steps by means of an example. Whilst in chapter 6 many qualitative remarks have been made about the capabilities and the limitations of workflow approaches, this chapter will place a relative value on these capabilities and limitations by taking into account the frequency with which exceptions occur, the cost of manually handling exceptions, etc. Ideally this sequence of steps should have been tested by the collaborating software vendor Swan Software, but insufficient time was available to do this. Thus the author conducted a paper-based analysis of possible application scenarios that can occur in the SOP domain.

2.0 Issues in defining sales order workflows properties

In order to quantify the capabilities and limitations of a given workflow approach applied to a sales order processing system in an appropriate way, a number of issues have to be taken into account:

1. *The type of company.* This can be defined with reference to one of the following four categories. 1: Make-To-Stock (MTS); 2. Assemble-To-Order (ATO); 3. Make-To-Order (MTO); 4. Engineer-To-Order (ETO) [Hei93, Lus93, Bro96, Kra98]. Existing workflow approaches and the new workflow approach may be less suitable to Make-To-Order and Engineer-To-Order company types than for Make-To-Stock and Assemble-To-Order. For MTO and ETO company types, sales orders are actually contracts where the main focus is on defining appropriate conditions of delivery, costs, etc. In these contracts the specific items, quantities and delivery dates are not defined.
2. *The general type of a sales order workflow.* Different types of sales order may exist, e.g. (1) 'a proforma invoice' workflow type, where first an invoice is printed before any deliveries are made or (2) 'a warehouse order' workflow type, where goods can be obtained directly from the warehouse without any need to send an order confirmation or the need to allocate items, etc. Thus the workflow type determines the activities that need to be included in a workflow specification (e.g. 'Enter Sales Order', 'Allocate Stock', 'Check Creditworthiness Customer', 'Pick Items', 'Enter Deliveries', 'Invoice Customer'). The applicability of a given workflow approach is determined by the extent to which these activities are performed in a distributed setting.

3. *The specific type of a sales order workflow.* There may be a need to specify different workflow types depending on properties of some entities that are processed in a sales order. Firstly activities may be performed conditionally depending on properties of the entity. For example it may not be necessary to perform the activity 'Check Creditworthiness' for selected items or customers. Secondly the order in which activities are executed may be different. For example for certain types of items it may be possible to immediately send an order confirmation to the customer instead of first allocating stock (as demand can be met in various ways).
4. *The type of exceptions that may occur in a sales order workflow.* For some companies exceptions may not be permitted, e.g. a customer may not be allowed to change the order quantity at a late stage because constraint conditions apply to the order. However, as operating policies and procedures change over time heavily constrained workflow approaches may prove severely limited. Thus in general companies should take into consideration whether requirements may change over time.
5. *The frequency with which exceptions may occur.* The frequency with which certain exception types occur may favor the adoption of certain workflow approaches over others. If certain types of exceptions do not occur often, then the use of an advanced type of workflow approach might not be warranted.

3.0 Measuring the capabilities and limitations of workflow approaches

The capabilities and limitations of existing workflow approaches and the new workflow approach will be examined with reference to a single measure termed 'manual corrective action'. This measure defines the number of corrective actions workflow system administrators have to do manually outside the control of a workflow system, namely in situations where a workflow system does not propose an appropriate action. Here it assumed that the 'manual corrective action' measure can be utilised by companies as a surrogate measure of the quality, speed and cost with which sales order workflows can be processed.

As discussed in section 3.0 of chapter 1 a required capability of a workflow system is that it exercises a tight control over those activities that need to be performed (by scheduling work, proposing actions to users and tracking the progress of work). However in complex application scenarios workflow systems may propose inappropriate actions and constrain the work of sales order processors. As a consequence (A) a customer may not accept a sales order (if the actions that the workflow system proposes are taken as a given) or (B) sales order processors may be forced to set up and undertake additional administrative actions to record necessary work actions. Both cases (A) and (B) are not ideal. Case (A) may result in the fact that a workflow system is abandoned after a period of time and the former manual system may be re-instated. However some workflow systems include capabilities to temporarily suspend the execution of a specific workflow instance and thereby to manually adjust the state of a workflow, where after workflow system control can be re-instated. Manual correction of the state of a workflow generally requires corrective and administrative actions on behalf of sales order processors and the workflow system administrator. Sales order processors will have to perform necessary compensation activities and report the occurrence and results of these activities to workflow system administrators. Generally workflow system administrator will need to update the state of the workflow. Yet there are risks in so doing. Sales order processors may forget to perform required activities, perform them in the wrong order or use the wrong parameters. Workflow system administrator may find it difficult to map performed activities onto the sales order workflow state and the workflow system administrator may also make similar kinds of mistakes.

The quality with which sales orders are processed may suffer because of mistakes made when adjusting the state of a workflow. Inconsistencies associated with workflow states may become clear immediately or they may come to light very late in the process. Once a workflow has been re-activated sales order processors may immediately detect the occurrence of mistakes. On the other hand inconsistencies that will become clear late on in sales order process may significantly impair the effectiveness and efficiency with which SOP operations are carried out. When mistakes are detected late, the correction of mistakes may require a significant, costly and time consuming effort.

The importance of processing sales orders in a timely way should be fully recognised. One important requirement is that due dates of sales orders are met. Another important requirement is that when an exception occurs its possible impact on many SOP plans should become clear as soon as possible. This is because different ways of handling exceptions can still be devised whilst at a later stage many plans have become fixed. Uncertainty left un-addressed may result in late, ineffective and costly actions. If some types of exception cannot be detected by a workflow system or if exception handling procedures prove not to be completely realistic, the workflow state must be corrected outside the workflow system control and much time may have passed before the new satisfactory state is reached to enable further processing. In effect the workflow system administrator can become a bottleneck with respect to processing exceptions, particularly when many exceptions occur at the same time. Subsequently it may not be clear which exceptions still need corrective action to be performed. Thus use of an inappropriate workflow system may actually decrease the speed with which exceptions can be handled. Naturally a company may, as a policy, decide to limit the number of exceptions that will be handled and as a consequence a workflow system may impact significantly on the way business is done.

Therefore it should be clear to the reader that generally various costs will be associated with the deployment of workflow systems. Firstly costs arise because of the need for corrective actions that a workflow system administrator has to perform. Secondly there are the costs of handling a situation where a workflow is not processed correctly because of mistakes made by sales order processors or workflow system developers (as discussed under the aspect of quality). Thirdly costs arise because of not being able to handle a given number of exceptions within a given period (as discussed under the aspect of timeliness).

4.0 Decisions at the base of the example sales order workflow

The example chosen in this chapter is based on:

1. Sales order processing in an Assemble-To-Order (ATO) company. This kind of company was selected as opposed to an ETO or MTO type of business because (A) a relatively high number of workflows are processed (e.g. hundreds to thousands per year), (B) the workflow duration is relatively long (e.g. weeks to months), (C) many SOP activities are performed in different departments (e.g. sales, finance, stock control) and (D) different exceptions may occur at various stages of processing.

2. The general sales order workflow type that has been selected is the 'standard' one which has three processing stages. In the first stage the order needs to be entered, component items need to be allocated, and the creditworthiness of the customer should be assessed. At the second stage the components will need to be picked and delivered to the customer. In the final stage the customer will be invoiced. However in this study focus has been on the first processing stage comprising workflow activities 'Enter Sales Order', 'Allocate Stock', 'Check Creditworthiness Customer' and 'Send Order Confirmation'. By focusing on the first processing stage it is possible to find sufficient detail to illustrate differences between various workflow approaches, whilst limiting the effort and time required to complete this activity. It is important to note that these activities are normally executed in different departments.

3. In the example only one specific workflow type will be considered that is capable of processing many common items. It is aimed at speedy processing of sales orders whilst ensuring that important criteria have been met. After the sales order has been entered, two critical activities 'allocate stock' and 'check creditworthiness customer' can be performed in parallel. If an order confirmation is to be sent, both activities have to complete successfully. It is possible to define other specific workflow types, for example for inexpensive items it might prove beneficial to skip steps like 'check creditworthiness customer'. However this study aims to outline relevant steps instead of defining all workflow types for a given company.

4. For the specific workflow type chosen under point (3) most common exceptions were considered such as 'order quantity/due date changes', 'stock allocation failures', etc. Less well-known exceptions such as price changes and MPS changes were not be considered in this example. In reality a company will also have to make some decisions about whether it is still

worthwhile to model all exceptions as usually such a modelling and calculation effort will need to be justified.

As no specific company data was readily available, it was decided that the number of typical exceptions would be determined from three factors: (A) the total number of workflow instances, (B) the relative frequency with which exceptions may occur and (C) assumptions about the number of times well-known exceptions will occur. E.g. assume that in one year 5000 workflows will have been processed. Also assume that three types of exception may occur and that their relative frequency is: exception A: 1; exception B: 1/10; exception C: 1/20. Finally assume that exception A has occurred 100 times in one year. Then exception B has occurred 10 times and exception C 5 times.

5.0 Common data flows found in an idealised sales order processing domain

In an Assemble-To-Order (ATO) company sales order workflows will comprise typical activities illustrated by Figure 80:

1. 'Enter Sales Order'. This is an information 'processing' activity the main purpose of which is to store data on customer orders and thereby to facilitate use of this data to other activities. Typically also the 'Enter Sales Order' activity will calculate the total order value, by making reference to the configuration of the order and relevant prices of those elements.
2. 'Allocate Stock'[43]. The purpose of this activity is to allocate components, defined as being available by the MPS, to a sales order according to quantities and due dates required by each customer. Components are not physically assigned to the actual order, rather generally they will remain as stock items until the order is typically picked or assembled.

---

[43] Some sales order processing applications include support for activities that incorporate 'Enter Sales Order' and 'Allocate Stock' activities as a single activity. However it may be that these two activity types are performed independently. This will particularly be the case if they are performed at different locations. For example a salesman may take an order at a customer location but will be "temporarily disconnected" from the main information system.

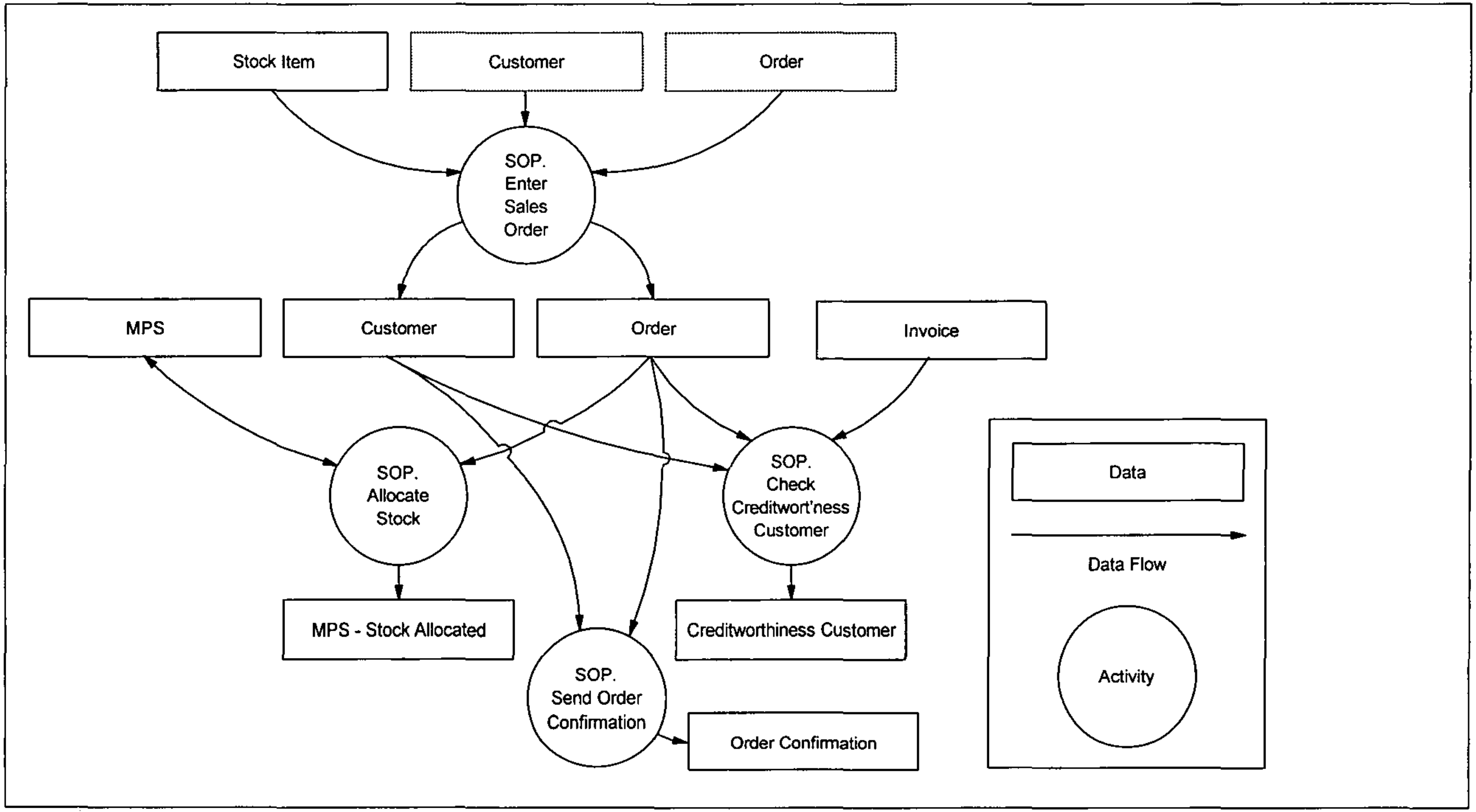


Figure 80: General data flow in sales order processing.

3. 'Check Creditworthiness of Customer'. The purpose of this activity is to estimate and/or predict whether a given customer is likely to fulfil its financial obligations. Typically the tasks involved need to take into account the customer's level of credit, the value of the current sales order and the value of other sales orders that remain to be invoiced, and outstanding debits for orders previously invoiced but not yet paid. The outcome of this activity is a 'go' or 'no go' decision based on the magnitude of a positive or negative number indicating a financial surplus or shortage.
4. 'Send Order Confirmation'. The purpose of this sales order processing activity is to report on properties of the order to the customer that confirms its order. These receipts are usually transmitted by post or electronically.

6.0 Common control flows found in an idealised sales order processing domain

Generally the order in which workflow activities are performed in sales order processing systems is defined by precedence relationships. This is illustrated by Figure 81. If no exceptions occur the flow of control will follow this general order. However if exceptions do occur the actual flow of control may be different in order to compensate for exceptions. Before any consumer activity can be performed producer activity 'Enter Sales Order' must have been performed.

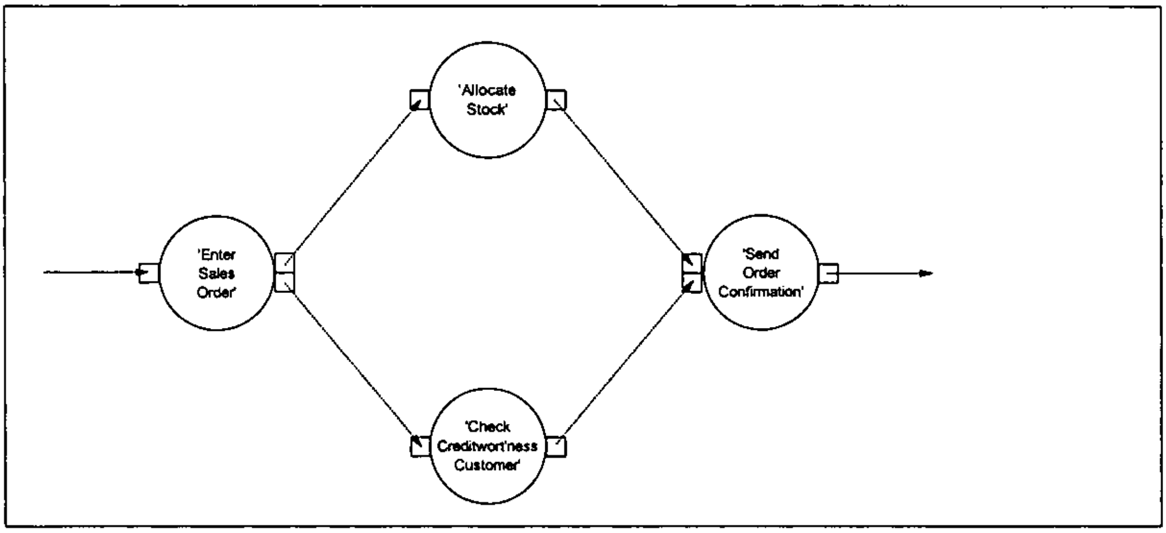


Figure 81: Precedence relationships governing general control flows in a typical sales order. workflow.

Also before consumer activity 'Send Order Confirmation' can be performed (1) a check should be made to determine whether enough actual components are available by completing an 'Allocate Stock' activity, and (2) a check should be made to determine whether a customer is creditworthy using preceding 'Check Creditworthiness of Customer' activity. Otherwise risk is taken that an order with a customer may not be satisfied. Typically activities 'Allocate Stock' and 'Check Creditworthiness of Customer' can be performed in parallel in order to save time.

7.0 Exceptions that arise in an idealised sales order processing domain

This section considers common properties of exceptions that arise in the sales order processing domain following either the occurrence of logical activity failures or the occurrence of exceptions arising from changes to the input of completed activities.

7.1 'Logical activity failure' type of exceptions

A logical failure exception will occur should necessary post conditions not be met following the completion of a SOP activity. The 'Allocate Stock' and 'Check Creditworthiness of Customer' types of SOP activity can fail logically while the other SOP activities listed in section 4.0 cannot fail in this way. Reasons for this are explained as follows:

1. 'Enter Sales Order'. This SOP activity type only processes a sales order from a customer and stores data related to sales orders. No decisions are made as a direct consequence of this SOP activity type as it is essentially a data entry activity. It follows that the execution of this activity cannot fail logically.
2. 'Allocate Stock'. Upon completion of this type of SOP activity the following post-condition should be met. For all sales elements specified by the sales order it should prove possible to allocate actual stock listed by the master production schedule before the required due date. Therefore if, for example, the master production schedule has already allocated much of the available stock to existing orders it may not prove possible to assign enough components to a new sales order so that a required due date can be met. Consequently execution of this SOP activity type can fail logically.
3. 'Check Creditworthiness Customer'. The execution of this activity will fail logically if the total outstanding order value is greater than the credit level of the customer.
4. 'Send Order Confirmation'. This type of SOP activity simply extracts appropriate data from the SOP system and forwards it to a customer. Consequently this SOP activity type cannot fail logically.

7.2 'Change to input' type of exceptions

If an input to a SOP activity changes it may or may not impact on the output of that or other SOP activities. This observation is illustrated below.

1. 'Enter Sales Order'. If only a price change occurs for any stock item the total order value of a sales order will probably still be correct because a promise/contract has been made to the

customer when the sales order was entered. However if a change occurs with respect to either the sales order quantity, due date or order configuration it is likely that the SOP system should treat this as an exception because the total order value will not be up-to-date. Hence the following table exemplifies the likely effect on the output of an Enter Sales Order activity when common types of data input change occur.

Data input change type	Effect on the output of an 'Enter Sales Order' activity
Sales order # quantity, due date, or configuration	Total order value is unlikely to be up-to-date. Also the database contains outdated values for the attributes quantity, due date and configuration.
Stock item # price	No effect expected on the output of the activity.

2. 'Allocate Stock'. The following table lists the effect of common data input changes on the output of an 'Allocate Stock' SOP activity.

Data input change type	Effect on the output of an 'Allocate Stock' activity
Sales order # due date	1. If the new due date is further into the future than the original due date, then actual components will be kept in stock and allocated to an order over the length of this period. Dependent on the length of this period this may constitute a significant increase in resources being tied up as allocated stock. 2. If the new due date is before the original due date, some of the actual components required may not be available on time. In either of these cases the result of the output of the 'Allocate Stock' SOP activity can remain valid, become partly invalid or even completely invalid.
Sales order # quantity	If the new number of ordered items is less than the original number of ordered items then too many components will be allocated to the order concerned. Where the new number is larger than the original number, then too few components will be allocated. Thus in either case the result will not be completely valid nor usually will it be completely invalid.
Sales order #	1. If an ordered item is removed from the order configuration,

configuration	a real stock of unwanted components will be allocated. 2. If a new order item is assigned is assigned to an order, no real stock will be allocated in respect of this item. Thus in either case the result will not be completely valid nor usually will it be completely invalid.

3. 'Check Creditworthiness Customer'. The following table lists the expected effect on the output of any 'Check Creditworthiness Customer' SOP activity following different types of data input change.

Data input change type	Effect on the output of a 'Check Creditworthiness Customer' activity
Sales order # quantity	If the new number of ordered items is less than the original number of items the customer will remain creditworthy; but in situations where the new number is larger than the original number a customer may no longer be creditworthy.
Sales order # configuration	1. If an item is removed from an order configuration, the customer will remain creditworthy, and 2. If a new ordered item is assigned to an order, a customer may no longer be creditworthy.

4. 'Send Order Confirmation'. The following table gives the likely effect on the output of a 'Send Order Confirmation' activity because of various types of change to its input.

Data input change type	Effect on the output of a 'Send Order Confirmation' activity
Sales order # quantity, due date, configuration	The original order confirmation sent to the customer will no longer be valid.

8.0 Possible states when exceptions occur in the idealised sales order processing domain

8.1 Possible SOP states when change to input exceptions occur

An analysis showed that a sales order workflow can be in one of five different states when an exception arises as a consequence of a data input change related to a sales order. Table 13 and Figure 82 have been constructed to illustrate this observation. The five possible states were identified by analysing the allowable order in which activities can be executed. This order is characterised by Figure 81. When a workflow instance is first created because of precedence relationships described previously only the activity 'Enter Sales Order' can be performed. This initial state is denoted as workflow state 1. Following completion of the 'Enter Sales Order' activity either an 'Allocate Stock' or a 'Check Creditworthiness of Customer' activity can be carried out, leading to workflow states 2 and 3. When one of these activities is complete the other should also be completed, leading to state 4. Subsequently a 'Send Order Confirmation' activity can be executed, resulting in state 5.

Workflow state	**Activities performed**
1	'Enter Sales Order'
2	'Enter Sales Order', 'Allocate Stock'
3	'Enter Sales Order', 'Check Creditworthiness Customer'
4	'Enter Sales Order', 'Allocate Stock', 'Check Creditworthiness Customer'
5	'Enter Sales Order', 'Allocate Stock', 'Check Creditworthiness Customer', 'Send Order Confirmation'

Table 13: States of a sales order workflow.

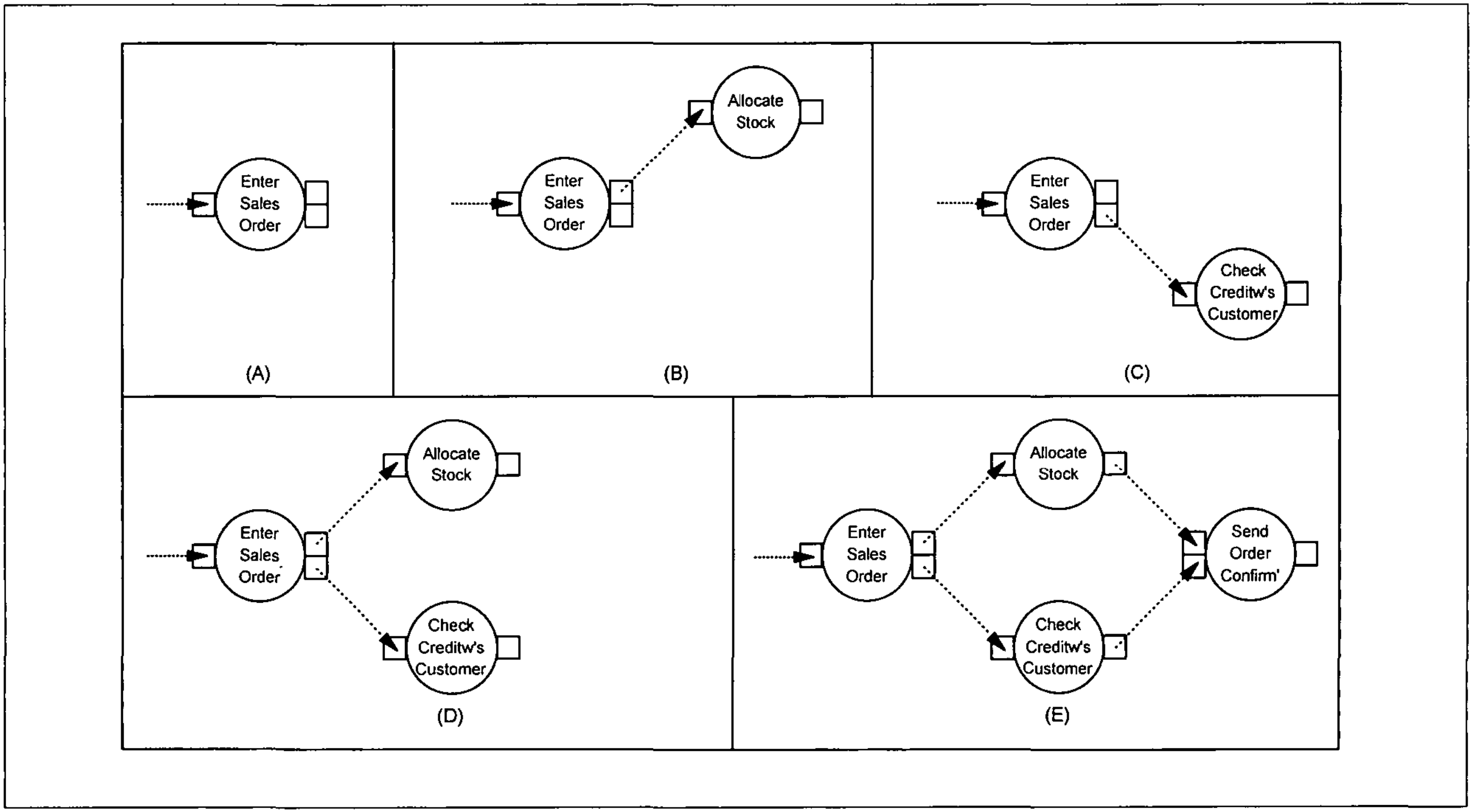


Figure 82: States of a sales order workflow.

8.2 Possible SOP states when logical activity failure type exceptions arise

There are two activities that can fail logically in the idealised sales order processing domain characterised in section 4.0, namely: 'Allocate Stock' and 'Check Creditworthiness Customer'. It follows that the possible states that a sales order workflow can reach when the activity 'Allocate Stock' fails logically will be those illustrated by Figure 83. If the 'Allocate Stock' activity fails logically then the 'Enter Sales Order' activity must have previously been performed to provide prerequisite inputs. However when such a situation arises the 'Check Creditworthiness of Customer' activity may or may have not been performed dependant on the progress of the sales order workflow.

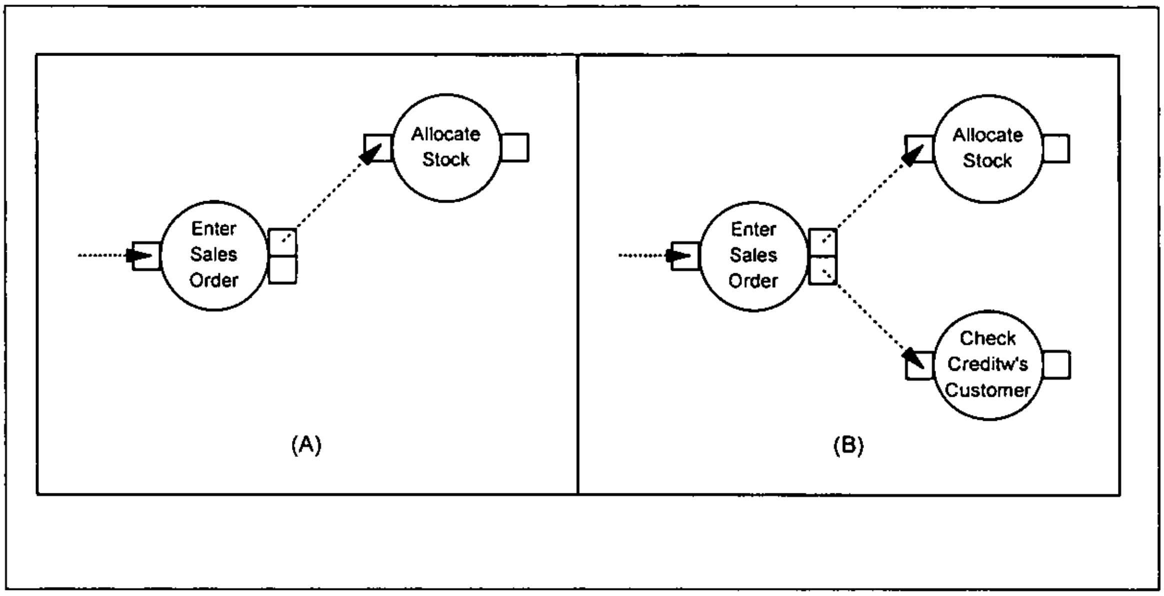


Figure 83: Possible workflow states when a logical failure of the 'Allocate Stock' activity occurs.

A similar argument holds true on considering the possible states of the workflow under conditions where logical failure of a 'Check Creditworthiness Customer' activity can occur, see Figure 84.

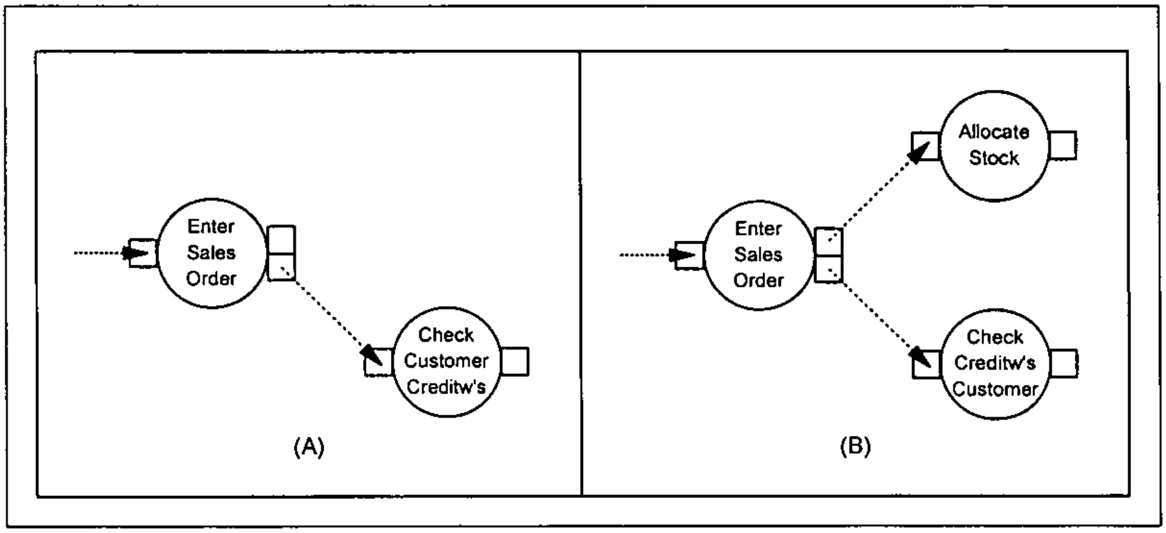


Figure 84: Possible workflow states when a logical failure of the 'Check Creditworthiness Customer' activity occurs.

9.0 Required compensation activity for a given exception type and workflow state.

When an exception occurs it is possible to choose from different options as follows:

1. Logical failure of the activity 'Allocate Stock'. Firstly it is possible to increase available quantities encoded by the MPS or to raise a special purchase or production order to meet extra demand. Yet these types of action will normally cross workflow boundaries. Secondly the delivery can be made at a later date, so that during the period of delay it becomes possible to allocate shortfalls in quantities from later MPS-periods. Thirdly the customer can accept the quantity that is actually available. Technically options two and three require compensation activities 'Modify Sales Order' and 'Re-Allocate Stock' to be performed (see Table 14 row I). If activity 'Check Creditworthiness Customer' has been performed, it must also be redone (see Table 14 row II).
2. Input change for the 'Enter Sales Order' activity. To resolve this type of exception it is important to check whether sufficient items can still be allocated to the order and whether the customer is still creditworthy. For example it may be important to establish whether sufficient stock can be allocated from earlier MPS periods if the due date is brought forward. Depending on the actual progress of the sales order workflow different actions may be taken (see Table 14 row III–VI).

Situation	Exception	State	Sequence of compensation activities	Temporarily block activities
I	A	1	'Modify Sales Order' → 'Re-Allocate Stock'	'Check Creditworthiness Customer'
II	B	2	'Modify Sales Order' → 'Re-Allocate Stock' + 'Check Creditworthiness Customer'	'Send Order Confirmation'
III	C	3	'Modify Sales Order'	'Allocate Stock' + 'Check Creditworthiness Customer'
IV	D	4	'Modify Sales Order' → 'Re-Allocate Stock'	'Check Creditworthiness Customer'
V	E	5	'Modify Sales Order' → 'Check Creditworthiness Customer'	'Allocate Stock'
VI	F	6	'Modify Sales Order' → 'Re-Allocate Stock' + 'Check Creditworthiness Customer'	'Send Order Confirmation'
VII	G	7	'Modify Sales Order' → 'Check Creditworthiness Customer'	'Allocate Stock'
VIII	H	8	'Modify Sales Order' → 'Re-Allocate Stock' + 'Check Creditworthiness Customer'	'Send Order Confirmation'

Table 14: Logical sequences of compensation activities.

3. Logical failure of the activity 'Check Creditworthiness Customer'. In this case firstly the sales order can simply be blocked until outstanding payments have been made. Yet this solution will not normally cross workflow boundaries. Secondly the customer may choose to order the quantity for which the customer is still creditworthy. Depending on the progress of the sales order workflow different actions should be taken (see Table 14 row VII and VIII).

It should be noted that new exceptions may occur while sequences of compensation activities are executed. Even though appropriate decisions may have been made at an earlier stage the state of a workflow may change because of unforeseen events. For example other sales orders may be returned from customers, stock corrections may be performed, etc. Thus whilst executing sequences of compensation activities new problems may arise and it may be necessary to undo the effects of some compensation activities.

In section 4.0 of this chapter it was decided that the number of typical exceptions would be determined from three factors: (A) the total number of workflow instances, (B) the relative frequency with which exceptions may occur and (C) assumptions about the number of times well-known exceptions will occur. For this workflow type it has been assumed that there will be 10000 instances processed per year. The relative frequency with which exceptions occur has been assumed to be (A) logical failure of activity 'Allocate Stock': 1/5, (B) input change at 'Enter Sales Order': 1 and (C) logical failure of activity 'Check Creditworthiness Customer': 1/10. A logical failure of activity 'Allocate Stock' for this workflow instance occurs 100 times a year.

Case	**No. of exc.**	**Trad. workflow approaches**		**ECA workflow approaches**		**Trans. workflow approaches**		**New workflow approach**	
		Corr. Action	Total Corr. A.	Corr. Action	Total Corr. A.	Corr. Action	Total Corr. A.	Corr. Action	Total Corr. A.
I	20	-	-	-	-	-	-	-	-
II	20	-	-	-	-	-	-	-	-
III	100	1	100	-	-	1	100	-	-
IV	100	2	200	-	-	2	200	-	-
V	100	2	200	-	-	2	200	-	-
VI	100	3	300	-	-	3	300	-	-
VII	10	-	-	-	-	-	-	-	-
VIII	10	-	-	-	-	-	-	-	-
Total			**800**		-		**800**		-

Table 15: Total number of corrective actions required.

10.0 Validation of research

Section 3.5.5 provides a rationale for validating findings and results of this study by means of a case study research approach. This section develops that argument further by considering: (i) general characteristics of case study research, (ii) specifics about case study that relate to the context of this study, (iii) how identified research findings and results could be validated and (iv) the applicability of data collection techniques in support of identified case study work and (v) a general consideration of risks associated with doing case study research.

10.1 General characteristics of case study research

A case study examines a phenomenon in its natural setting, employing multiple methods of data collection to gather information from one or a few entities (people, groups or organisations). The boundaries of the phenomenon are not clearly evident at the outset of the research and no experimental control or manipulation is used [Ben84, Ben87, Bon85, Sto78, Yin84]. Key characteristics of case studies are:

1. Phenomenon is examined in a natural setting.
2. Data are collected by multiple means.
3. One or few entities (person, group or organisation) are examined.
4. The complexity of the unit is studied intensively.
5. Case studies are more suitable for the exploration, classification and hypothesis development stages of the knowledge building process; the investigator should have a receptive attitude towards exploration.
6. No experimental controls or manipulation are involved.
7. The investigator may not specify the set of independent and dependent variables in advance.
8. The results derived depend heavily on the integrative powers of the investigator.
9. Changes in site selection and data collection methods could take place as the investigator develops new hypotheses.
10. Case research is useful in the study of "why" and "how" questions because these deal with operational links to be traced over time rather than with frequency or incidence.
11. The focus is on contemporary events.

An important advantage of case study research over other methods of research is the possibility to observe a phenomenon in its natural setting, without interfering in its results. This allows new hypothesis to be explored and developed that can be validated at the end of the process. However the lack of skills of researchers in carrying out a case study may result in a sloppy investigation. Also a researcher may be biased in the directions on first findings and conclusions.

Data may be collected in the following ways:

1. Interviews. Interviewing of staff and managers will quickly give a good understanding of the validity of many of the key assumptions made in a study.
2. Direct observation. Direct observation is a data collection technique where contemporary events in the real system are recorded and analysed over a given period. If the period is chosen sufficiently long, it is possible to obtain a set of events that may prove the key assumptions made or that may refute them.
3. Archival records and documentation. Written material ranging from memoranda's, reports, meetings notes, etc. may be useful sources of evidence. Other types of sources may be automated information systems. To filter appropriate information from the former source a lot of time may be taken since a company may have gathered large amounts of paper over time. To gain information from the latter source it is necessary to have an understanding of the system that stores it.

10.2 Specific use of case study research within the context of this study

Yin explains that case study research offers an important way of understanding research phenomena in the area of information system design and use [Yin84]. However the collaboration with software vendor Swan Software as well as on the experience of the author during early stages of research has an impact on the use of this research method within the context of this specific study. In general case studies are more suitable for the exploration, classification and hypothesis development stages of the knowledge building process. However the collaborator provided the initial focus of this research as the problems faced by personnel at Swan largely determined the motivation and context of research (see section 3.1 of chapter 3). Indeed the collaborator has already performed part of the exploration, classification and hypothesis development stages of the knowledge building process. Indeed case study requirements 5 and 7 listed in the previous section were found to be difficult to meet during this study. With respect to point 7 the author observed many possible independent and dependent variables in advance of hypotheses development.

Thus the use of case study approach will largely focus on the validation of the problems and issues as identified by Swan Software as well as further work done by the author.

10.3 Research results to be validated

This study generated research results and findings that require validation by suitable means. Some findings were by nature high-level and abstract (such as the new workflow model for sales order processing), others more concrete (such as developed requirements descriptions for sales order processors and workflow system developers).

A list follows of some of the main research findings and results that required validation.

1. Identified 'unsolved requirements' with reference to the management of common workflows found in sales order processing domains (see section 3.2 of chapter 1).
2. A new workflow model developed to describe key characteristic properties of sales order processing (see section 5.2 of chapter 5).
3. Requirements descriptions for sales order processors and workflow system developers (see section 5.3 of chapter 5).
4. An analysis of the capabilities of existing approaches to workflow management (chapter 6).
5. The specification of a novel workflow approach for use in sales order processing domains (see chapter 7).
6. The design of a component-based workflow system for use in sales order processing domains (chapter 8).

10.4 Applicability of data collection techniques

This section considers the general suitability of data collection techniques. The techniques listed were utilised in this study to determine the validity of research statements made in this thesis about sales order processor and workflow system developer issues.

Research results	**Interviewing**	**Observation**	**Archives**
Result 1	High	Low	Low
Result 2	Medium	High	High
Result 3	High	High	Medium
Result 4	High	High	Medium
Result 5	Medium	Low	Low
Result 6	Medium	Low	Low

Table 16: Applicability of data collection techniques for established research results.

The three data collection methods mentioned below can each be deployed in a coherent way as the basis of a more complete data collection technique, where each method has distinctive beneficial characteristics. In general, interviewing is suited to testing key assumptions, while observation will normally provide useful qualitative insights. Whilst the inspection of artifacts generated from many projects is likely to lead to quantitative measurement.

Potentially the interviewing of sales order processors and workflow system developers (i.e. staff and managers) can rapidly provide evidence in support of many of the key assumptions (i.e. result 1), sales order processor and workflow system developers requirements (i.e. result 3) and the capabilities of workflow approaches (i.e. result 4) that have been developed in this study. In today's generation of sales order processing systems human employees play an important role in deciding how to handle exceptions. Thus over the years experienced personnel fulfilling the role of sales order processors may have developed structured ways of working and those persons can provide valuable base knowledge for researchers and practitioners. Also over the years experienced developers and managers have gained key insights in the applicability of their workflow system technology. However more abstract models of sales order processing (i.e. result 2) as well as new models of workflow (results 5 and 6) may be more difficult to understand. For example theoretical issues, such as the extent to which responses to exceptions can be pre-specified or is to be determined at execution time, may not be fully appreciated by practitioners.

The technique of observation is useful in that it can help to validate result (2) new workflow model for sales order processing, result (3) the requirements of sales order processors and result (4) the capabilities of the deployed workflow approach. Namely these results are directly related to the way sales order processors perform their work and this process can be observed and analysed by a researcher. However it is difficult to validate more abstract statements (result 1) and a new approach to workflow (results 5 and 6) by means of observation (in the context of a case study[44]). Also this data collection technique may be useful to confirm statements made during interviews as well as to observe issues that have not come up during interviews).

In the domain of sales order processing, information useful for research investigation may be archived by ERP-systems. Examples of information that is typically stored in ERP-systems are (A) sales orders that get cancelled; (B) stock-outs recorded automatically in the form of back orders; (C) distinction can be drawn between required and actual due dates, (D) reasons can be recorded when the amount invoiced does not match the amount paid (e.g. for cases of wrongly quoted prices). These kind of exception have been widely recognised key performance indicators in the operation of a SOP-system. By writing ad-hoc queries it is possible to retrieve this kind of information and to process that data in support of research theory building and testing. In particular it is possible to validate research results 2 with this data. Also companies may have documented some of their SOP requirements when acquiring their system and may identified some of its limitations in use (results 3 and 4). Research results 1, 5 and 6 are more difficult to back-up using archived material.

---

[44] See section discussion as well.

In the domain of workflow system developers, artefacts such as project documentation, memo's, meeting notes, program code, etc. may contain useful material to underpin assumptions. Whilst project documentation and program code may contain useful workflow specifications, memo's and meeting notes may reveal key issues in the development of workflow specifications.

10.5 Risks associated with doing case study research

This section discusses some of the risks inherent in doing case study research for different data collection techniques:

1. Interviews. However there are also some dangers associated with this approach:
    - Employees may have insufficient knowledge of key issues, possibly because they have just started the job, have fixed ways of working, are concerned only with a specific part of the process, or are concerned with process management using only an abstracted or simplified understanding. Generally employees should back-up their claims with the use of examples, or in other ways.
    - Sales order processor and workflow system developer personnel may not understand the questions formulated by the researcher. Employees must be able to grasp complex research concepts and their significance. Thus it is important to define terms used during interview questions in order to minimise the chance of misunderstandings between the interviewer and interviewee. If they cannot, the interview may not be constructive and may not yield useful results.
    - Results from initial interviews may influence observations made at a later stage as employees are known to optimise issues brought forward during interviews.
2. The technique of observation is useful. However in this study the following limitations associated with the use of this technique were observed:
    - In operational systems there are always time and cost constraints available and it may not always be possible to explain properties of events in an operational context.
    - Sales order processor and workflow system developer personnel may fear the conclusions of the research and will in some cases not co-operate with researchers and share information freely.
    - Observations made by developers and managers will be consequent upon a subjective task performed by the researcher who may be inclined to map recorded events to already developed assumptions.
3. Archival records and documentation. Specific limitations with respect to this data collection technique can be defined for each type of user:

- Workflow system developers. Whilst project documentation of workflow system developers may be open to different interpretations (because it may be ill-defined in a natural language), their program code may be difficult to understand.
- Sales order processors. ERP-systems may only record certain types of exceptions.

10.6 Use of further research methods

In the previous section a number of risks associated with doing case study research were presented. Yet case study research has some other broader limitations:

- Case study research is naturally limited to one or a few companies and results may be difficult to generalise across various businesses and industries. In this respect survey-based research can provide an outcome. Survey research is a method of systematically collecting information about a set of cases (such as people, objects) by asking pre-formulated, pre-sequenced questions to a sample of individuals drawn so as to be representative of a defined population.
- The phenomenon is examined in a natural setting and no experimental controls or manipulation are involved. This limits the extend to which the capabilities and limitations of the new workflow approach (as well as the component-based workflow system) can be assessed. In this respect laboratory experiments can be used to test the component-based workflow system; ERP and workflow software may be in-stalled on computers in a lab and sales order processors may be invited for a day of testing. Test scenarios in which exceptions occur will be put in front of sales order processors and they may be asked to solve these situations. Experienced sales order processing and workflow system developer personnel must be able to determine the applicability of the proposed workflow approach on its main ideas. In these type of experiments it is possible to exercise a great deal of control and discuss, analyse and define various business process examples, company and industry definitions where the proposed approach may be deployed.

11.0 Discussion

In this chapter many issues have been identified that are of relevance when quantifying the capabilities and limitations of various workflow approaches. Even though in chapter 6 many statements have been made about the qualities of different workflow approaches, on some occasions it may be necessary to assess the quantitative significance of workflow features relative to each other. First of all it was noted that the type of industry or business that a company operates in, may largely determine the applicability of a workflow approach. It was proposed that ETO and MTO type of companies may be less suitable to the new workflow approach than for example MTS and ATO types. Other important issues in evaluating workflow approaches were found to be general & specific types of workflows and the type of exceptions that may occur in a sales order workflow. By using these criteria a company can quickly identify workflow types of relevance to their business. Otherwise a company may focus on a certain aspect (e.g. on handling certain exceptions), whilst not addressing other aspects (e.g. not realising that the workflow approaches are not suited to their operating environment).

In comparing the new workflow approach with respect to previously existing workflow approaches it was decided to focus on the need to perform corrective action outside the control of the workflow system and from that perspective to consider when a workflow system cannot support certain application scenarios. This measure is of great importance if a company is to deploy a workflow system in a commercial environment since there are various risks associated with circumventing the control of a workflow system. For example after some time it may be decided that a workflow system is more of a hindrance than a support, therefore requiring a company to abandon the automated workflow system and possibly to return to some original manual system. Yet other companies may decide to use other measures to determine the applicability of a workflow approach (e.g. focus primarily of the speedy processing of sales orders). These measures must be linked to logical scenarios of compensation as defined in Table 14.

This study, however, has cut some corners when defining a complete example. Firstly the study has only focused on the first processing stage of a 'standard' workflow type comprising workflow activities 'Enter Sales Order', 'Allocate Stock', 'Check Creditworthiness Customer' and 'Send Order Confirmation'. As this a largely a data processing application it may favour transactional workflow approaches and the new workflow approach when compared to others. If the comparison of workflow approaches defined in Table 14 of section 9.0 is to have practical relevance, activities like 'pick items', 'deliver items', 'invoice customer', etc. should be included. Secondly few types of exceptions have been defined and their impact evaluated. That said the exceptions that were defined (and their impact studied) are probably most important, but other exceptions should have been identified and discussed. Thirdly Table 14 shows the number of compensation activities that should be performed yet no definitions of supporting exception handling scenarios for previously existing workflow approaches was made. It may also be difficult to check whether the predicted data is correct. Fourthly the need to perform corrective action does not include activity scenarios where activities need to be blocked or where exception handling procedures set into motion need to be reversed. Thus some work still needs to be done to conclusively show that the new workflow approach is better than other pre-existing workflow approaches with respect to meeting sales order requirements for a specific workflow type. However it is strongly argued that many important steps forward have been specified and prototyped, and key related analytical results have been worked out.

It follows that there remains a need to further elaborate issues mentioned in the first paragraph of this section. For this to be achieved characteristics of ATO and MTS companies need to be defined in greater detail. Further all types of general workflow type (such as 'proforma invoice') will need to be identified and their properties well defined. This should facilitate definition of all types of exception that may occur when each activity is executed. When these issues have been analysed, companies should readily be able to identify those workflow approaches and systems that best suit their needs.

The preceding paragraph explains how analytical methods and results prototyped by this PhD study can be developed into methods (and possibly exemplar reference models) that industry can deploy when selecting a suitable workflow approach and system. That said, complicating practical issues may mean that specific companies still find it difficult to quantify benefits and risks associated with different choices. This will necessitate simplification to be made. For example a great deal of effort and time may be required to identify all possible workflow states, exception types and required compensation activities. This will particularly be true if a company has many different types of workflow. Such a situation may warrant a focus on a limited number of workflow states and exception types. Yet it is important to note that the focal workflow states and exception types should be representative of other states and types.

10. Research review

1.0 Review of research methodology

Section 6.0 of chapter 3 describes a research methodology that was determined to achieve the research aims outlined in section 4.0 of chapter 3. The following phases of research activity were included:

1. Detailed literature review.
2. Initial modelling and classification of sales order processing exceptions and their impact.
3. Sales order processing domain analysis, to determine the extent to which previous workflow approaches support the hand-ling of exceptions.
4. Design and part-implementation of a new approach to robust workflow specification and change.
5. Case study evaluation of the capabilities and practicability of the new approach.

This section provides an overview of results achieved from performing these research activities. In so doing it draws a 'red line' through the thesis. Also discussed are new insights that have been gained and outstanding limitations of performed research activities.

Under research activity 1 the workflow literature was studied and analysed. Three kinds of approach were identified with respect to the specification and enactment of workflows. These approaches were classified as 'traditional', 'event-condition-action' and 'transactional' workflow approaches. Detailed study and characterisation of each approach revealed two typical ways used to handle exceptions, namely: (1) by pre-defining static sequences of compensation activities and (2) by dynamically generating some compensation activities from exception handling knowledge. Early practical work of the author showed that either of these established ways of handling exceptions introduced limitations on the capabilities and applicability of current workflow approaches. The former way does allow a workflow system developer at the time of workflow design to pre-define unique sequences of compensation activities but only for some kinds of exception instance that may occur. The latter way requires knowledge to be abstracted about how certain sequences of compensation activities should be generated and used at run-time. Here it was observed that to be practical such an abstraction would need to explicitly encode application domain properties. However explicit importation of domain properties would likely significantly constrain cross-domain deployment. Although the review work carried out as part of this research activity has provided valuable new insights, time constraints did not permit other interesting research avenues to be pursued. Characteristic properties of the three types of workflow approach were analysed from different user viewpoints but their capabilities and applicabilities were not quantified and compared relative to each other. Secondly aspects of workflow specification re-use (as discussed in section 3.7 of chapter 2) could not be studied within available time constraints, even though this could have provided new understandings about managing workflows.

In research activity 2 for the sales order processing (SOP) domain, common exception were analysed and classified. Also key properties of SOP exception types were determined and formalised, as was their likely impact on SOP workflows taking due account of workflow states at the time exceptions occur. Here the effects of different exception types was explained in a more complete and formal way than previoiusly described in the literature, namely in terms of concepts like 'data flow' and 'control flow' dependencies. This led to the identification of new workflow specification and enactment requirements for two distinctive groups of 'workflow system user'. Later in research activity 4, these improved understandings about SOP exceptions made it possible to identify novel ways of (1) enabling sales order processers to enact more complete workflow specifications and (2) reduce the time and effort required by workflow system developers to specify, develop and change workflow specifications. Particular improvements to workflow specification and enactment were sought with a view to developing and enacting exception handling specifications. During research activity 2 it was also observed that previous literature reported on various ways of synchronising enterprise activities. Much of this literature has a computer science orientation and described candidate synchronisation mechanisms that could have useful application within future generations of practical workflow system. From an enactment point of view, where co-ordination actions need to be transmitted to real systems, some of these concepts were found to have particular relevance.

Having characterised SOP exception handling requirements, research activity 3 sought to determine the extent to which the three kinds of previously available workflow approach could meet both sales order processor and workflow system developer requirements. Previous literature had described in some detail aspects of the capabilities and limitations of workflow techniques. However, as a consequence of research activity 3, for the first time their capabilities and limitations were determined in a coherent fashion, with reference to both sales order processor and workflow system developer requirements. It was observed that previous approaches centred on developing static specifications for compensation activities (as is the case for 'traditional' and 'event-condition-action' workflow approaches) had capabilities to meet a subset of sales order processor requirements but gave little support for workflow system developers, in terms of reducing their effort and improving the flexibility and correctness of workflow specifications. On the other hand previous workflow approaches based on the dynamic generation of compensation activity (such as 'transactional' workflow approaches) had superior capabilities with respect to effort, flexibility and correctness concerns of workflow developers but they provide minimal support for completeness requirements. Essentially therefore research activity 3 provided a 'measured understanding' of the state-of-the-art (prior to this research study) of alternative SOP workflow approaches and their potential application in SOP domains.

Under research activity 4 a new workflow approach was conceived, specified and developed. It was decided that any approach which is wholely based on predefining static sequences of activity will remain limited because various SOP exceptions can occur when the workflow has reached different states of process, and therefore specific compensation activities need to be specified for each exception type and current workflow state. Therefore it was decided that the new approach to workflow specification and enactment should realise an improved way of dynamically generating compensation activities, in order that any missing semantics in pre-specifications are determined 'on the fly'. Therefore it was found to be necessary to pay particular attention to concepts associated with 'workflow state' and 'workflow transformation'. The encoding of multiple states of progress of workflows was made possible by conceiving, developing and using a *dual standard and exception workflow* concept. In addition an identified requirement to transform workflow states (by dynamically generating necessary compensation activities) was satisfied by realising a process of: detecting the occurrence of various kinds of exceptions; and tracking associated data and control flow dependencies, prior to dynamically generating specific exception workflow sequences.

In research activity 5 capabilities of the new workflow approach were examined with reference to previously identified exception types and their impacts in SOP domains. Also asssessment was made of the practicability of the new approach from sales order processor and workflow system developer viewpoints. The new approach was found to provide improved capability relative to previous approaches, when judged in terms of both sales order processor and workflow developer requirements.

2.0 Discussion

This section discusses observed limitations of the new workflow approach, its broader implications and its potential for further development. This is done from the perspective of both sales order processors and workflow system developers.

2.1 Discussion of sales order processor issues

Although the new workflow approach meets many of the SOP-requirements (as discussed in section 7.0 of chapter 7) further research is still required to prove its full application. The strength of the new workflow approach is that it handles exceptions by tracking the progress of a workflow using data and control flow dependencies. However this highly structured approach to handling exceptions may not always prove adequate and in some situations it may be necessary to override this behaviour. One example of such a situation corresponds to the case where a logical failure occurs but upstream planning activities need not be redone (discussed too in section 7.0 of chapter 7). A second example situation corresponds to the case where the results of an activity need not be cancelled immediately because they may be useful at a later stage. Thirdly under some conditions different precedence orders may be applicable amongst activities. If additional structures and services (that satisfactorily address these limitations) cannot be identified and developed to fit in with the new approach, then it may aprove that the new approach is less generally applicable than desired. Furthermore it must be acknowledged that the new approach only incorporates capabilities to detect and handle three types of exceptions, namely logical activity failures, input changes, and business rule changes. Yet there may be other exceptions types that occur in SOP-domains that cannot be adequately represented by these exception types, and thus may not fit effectively within the proposed framework.

A major, more broad, criticism of the new workflow approach is that exception workflows are not automatically optimised according to decision rules. In cases where a root exception occurs at an activity the proposed workflow approach will calculate derived exceptions to other workflow activities. First then the costs associated with handling the exception will become clear and the sales order processor may wish to modify some of the properties of the exception (possibly in co-operation with the customer) in order to obtain a better solution. Yet manual calculations are required to perfom 'what-if' analysis. Once a different approach has been chosen it may also be that the enactment plans may change. Indeed in essence the new workflow approach can be viewed as being a trial-and-error approach where SOP employees from different functions (such as sales, distribution and finance) give their input to an exception, but as yet they do not necessarily co-operate when determining the best scenario.

On the other hand the new workflow approach naturally supports the distribution of decision knowledge. Even though workflow system modellers may have a good understanding about the likely impact of exceptions on operations to be performed by sales order processors and they can model exception handling in some detail, in general it is probably that SOP employees will have a better, in-depth understanding of the operations they perform. A major advantage of the new workflow approach is that all distributed activities that have been affected by an exception will be identified. Next SOP-users must report appropriate changes to SOP plans because of exceptions that have occurred. An important advantage is that this can now be done at run-time when the precise properties of an exception, including the state of SOP plans, are known to the sales order processor. A workflow modeller can only make limited assumptions about possible states and determine the most likely applicable exception handling procedures in advance. A second advantage is that because sales order processors have an intimate knowledge of their operations they can conceive different solutions to a given problem. They are likely to be held accountable for their work and thus they may want to take on this kind of responsibility. Even though it is possible, in the case of some exception types and instances, to predefine the best way of achieving exception handling such as by taking into account established knowledge from all departments involved in SOP, such an approach will normally involve a significant modelling effort. In the proposed approach far less modelling effort is required. Further normally it will take time to train sales order processors about all aspects of activities and their likely impact on other functions, and therefore all employees cannot be expected to understand all aspects fully.

It is probable therefore that the new workflow approach has particular potential to support sales order processors working in complex application environments, because:

1. Multiple exceptions can be handled concurrently; for every exception an exception workflow can be started that records an alternative state of the workflow.
2. Conflict detection between exception workflows is now possible. In the new workflow apaproach, exception workflows record changes to activity outputs in the standard workflow and in general the exception workflows will be subsets of their standard workflow. It follows that if an exception

workflow records different outputs for the same activity in a standard workflow, then a conflict may exist. Therefore there arises a need for rules to compare different versions and determine whether they are consistent or in conflict. If they are in conflict the workflow system should propose alternatives. In this way conflicts can be detected at an early stage and therefore alternative scenarios can be developed at an early stage.

3. It is possible to evaluate the effects of multiple exception workflows in a coherent fashion. If exception workflows are combined into a super exception workflow, this kind of exception workflow will show the effects of multiple exception workflows in a unified way. Even though multiple exception workflows may not conflict a sales order processor may wish to view their effects in a combined fashion.

2.2 Discussion of workflow system developer issues

The new workflow approach meets many of the observed workflow system developer requirements. The new workflow approach is not based on an approach of explicitly predefining exception handling specifications (such as for example explicit sequences of compensation activity). Rather the new workflow approach handles exceptions by means of specialised rules that take into account the properties of exceptions and actual data and control flow dependencies amongst SOP activities in an executing workflow. In this way appropriate compensation activities can be generated at run-time taking into account different states of progress of a workflow and different properties of exceptions. Instead of pre-specifying similar exception handling procedures for different workflow states, the proposed approach requires only data and control flow specifications to be modelled and predefined.

However with the new workflow approach appropriate exception handling specifications still need to be developed to address current SOP limitations (as discussed in the previous section). Indeed the proposed workflow approach imposes a rigid framework on the way that SOP exceptions are handled (but under some conditions planning activities may not need to be redone). To overcome the outstanding limitations, novel exception handling specifications may need to be developed and this may again impact on the needed development effort and on the ease with which specifications are checked for correctness and modified.

The potential of the new workflow approach can be explained from another perspective, that of providing a more formalised approach to (1) workflow state definition and (2) specification of workflow transformations. In general it can be observed that approaches that pre-specify explicit compensation activity (such as 'traditional' and 'event-condition-action' workflow approaches) do not model state and also have limited capability to specify state transformations. 'Event-condition-action' workflow approaches have 'conditions' that refer to the state of a workflow; but the conditions that are defined do not necessarily encode all possible workflow states. Furthermore the limitations of 'event-condition-action' with respect to state transformation stem from two key issues. Firstly it is almost impossible to specify all needed exception handling scenarios, in particular for cases where new exceptions occur whilst already processing an exception. Therefore it will be difficult to anticipate the actions required to handle multiple exceptions.

Secondly developed exception handling procedures may in fact be quite similar even though part of them may not be reused. Because 'event-condition-action' approaches are based on the idea that procedures to handle exceptions can be predefined, complexity is likely to grow rapidly if a workflow has many activities, if many exceptions may occur and exceptions may occur whilst handling exceptions. In the new workflow approach, workflow state is defined in a structured manner in terms of standard and exception workflows. Furthermore workflow state transformation is achieved in a systematic way by taking into account data and control flow dependencies. Now it is possible to develop additional transformation rules for truly exceptional cases.

3.0 Contributions to new knowledge

This section reviews contributions to new knowledge made in the field of study.

Section 6.0 of chapter 3 reports on a set of research activities that sought to develop an improved understanding of exception handling in the domain of sales order processing. Consequent on these activities the following contributions were made:

1. A new analysis of exception types and their impact on the workflow state in the sales order processing domain was carried out. Three main types of exception were identified that characterise common SOP exception conditions that can arise. The three exception types were found to be as follows: 'input change', 'logical activity failure' and 'business rule change'. Sections 2.1 and 2.2 of chapter 5 respectively developed descriptions of generic properties of these exception types.

   Two typical ways in which any SOP workflow state is impacted on by these exception types were also identified and characterised. The impacts identified were as follows:

   - Data flow dependencies were found to pass on input change or logical activity failure exceptions at a given activity to derived exceptions to other activities (see section 2.1 of chapter 5).
   - Control flow dependencies were found to propagate changes to the state of business rules to the state of activities in a workflow (see section 2.2 of chapter 5).

2. A new requirements definition was developed from the perspectives of sales order processors and workflow system developers which characterised the SOP domain needed to handle exceptions adequately well. The developed definition comprised:

   - Sales order processor requirements expressed in terms of the completeness of workflow specification-related issues involved when synchronising the execution of workflow activities and exceptions (see section 3.1 of chapter 5).
   - Interrelated workflow developer requirements expressed in terms of the effort & flexibility associated with workflow development (see section 3.2 of chapter 5) and the correctness of workflow specifications (see section 3.3 of chapter 5). These requirements were formulated with reference to changes to the workflow specification.

3. State-of-the-art approaches to workflow specification and enactment (described in the literature and embedded into current generation workflow tools) were classified, contrasted and compared. Classification and comparison was based on a consideration of their characteristic modelling and enactment constructs. The workflow approaches reviewed were placed into following classes:

    - 'Traditional' workflow approaches, e.g. IDEF0/SADT, Grai Nets, IDEF3, IEM and CIM-OSA (see section 2.0 of chapter 4). Characteristic modelling concepts and mechanisms were linked to notions of (A) data flow, (B) control flow and (C) post-conditions.
    - 'Event-condition-action' workflow approaches, e.g. Rapide and WIDE (see section 3.0 of chapter 4). Event-condition-action workflow approaches were found to support properties of (A) event types, (B) condition types and (C) action types.
    - 'Transactional' workflow approaches, e.g. Sagas, Contracts, and Partial Rollbacks (see section 4.0 of chapter 4). Typical characterising concepts and mechanisms were found to be based on: (A) workflow interdependencies, (B) the scope of compensation, (C) type of compensation and (D) the order of compensation.

    With respect to these three classes of contemporary workflow approach, two fundamentally different techniques were observed as being used to handle exceptions (see section 4.5), namely (A) by developing static specifications of compensation activities and (B) by dynamically generating compensation activities.

4. A new capability assessment was developed with respect to state-of-the-art approaches to workflow (identified under (3)) with reference to sales order processor and workflow system developer requirements specified by the new research described under (2).

    It was found that:

    - None of the contemporary approaches to workflow engineering meet all completeness requirements of sales order processors (see section 2.0, 3.0 and 4.0 of chapter 6). Current means used to dynamically generate compensation activities do not adequately support the needs of realistic SOP scenarios subject to exception occurrences. While approaches that execute predefined sequences of compensation activities address only some of those requirements.
    - Contemporary workflow approaches that require responses to exceptions to be pre-programmed by means of explicit sequences of compensation activities (such as is required by traditional and event-condition-action workflow approaches) were found to (A) take great effort to develop, (B) are difficult to maintain and (C) are hard to check for correctness. On the other hand contemporary approaches that dynamically generate required compensation activities are relatively easy to develop, to modify and to check for correctness.

5. Subsequent research investigation uncovered new ideas and definitions related to 'workflow state' and 'workflow transformation'.

- It was observed that the state of a workflow can be characterised by the statuses of standard and exception workflows that are capable of capturing alternative versions of workflows needed to handle multiple exceptions (see section 3.0 of chapter 7).
- Two new and important concepts were observed as being able to underpin the transformation of a workflow. Firstly for every exception that occurs a different version of a standard workflow can be generated to characterise an exception workflow. Secondly, until an exception is accepted, the state of an exception workflow may still be subject to change (or may even need to be cancelled).

6. One study identified candidate structures and rules with capability to underpin the new concepts mentioned under (5).
    - This encompassed the definition of new exception workflow structures (see section 5 of chapter 7). A number of exception workflow modelling concepts and constructs were defined. These related to root exceptions, compensation solutions and derived exceptions. Further recommendations were made as to how these concepts and constructs may be implemented based on extensions to SADT/IDEF0 and IDEF3 specifications.
    - Formulated new rules to transform the state of standard and exception workflows (see section 6 of chapter 7). Transformation of the states of standard and exception workflows was proposed in two phases, namely in (A) an evaluation and (B) a completion phase (see section 6.1 of chapter 7 respectively 6.2). Furthermore concrete rules were defined to handle changes to (A) the flow of data and (B) the flow of control (as defined under (1)).
7. The study built and part-tested a prototype component-based implementation of the new workflow approach proposed (this proposal is outlined in Chapter 7). Specific contributions to knowledge made in this respect centred on:
    - A component interaction definition. In order for any workflow system component to synchronise its activities a number of event types needed to be characterised and means proposed to specify their characteristics (see section 2.0 of chapter 8).
    - Necessary component functionality enhancements needed to be conceived and specified and related extensions developed to handle exceptions:
        1. Application component. New functionality was developed to manage different versions of activity output (see section 3.3 of chapter 8).
        2. Agent component. Extra state definitions were developed for work items (i.e. sales orders) and functionality was provided to manage the transition of these states via the use of predefined state-transition diagrams (see section 3.2 of chapter 8).

3. Workflow engine component. Enhanced workflow state definitions (via the use of exception workflows) were developed and enhanced mechanisms prototyped (see section 3.1 of chapter 8).

Further the new logical approach to workflow proposed by this study was mapped onto existing workflow system components currently provided by software vendors.

4.0 Future research (within the scope of this research study)

Although a number of important contributions to new knowledge have been made, a number of study limitations were identified, which can be elaborated in future work. The following possibilities have been identified for future research directly within the scope of this research:

1. Further comparison of characteristics of the identified classes of contemporary workflow approach, namely traditional, event-condition-action and transactional workflow approaches. This study did characterise properties of (A) data flow, control flow and post-conditions modelling constructs by traditional approaches, (B) event types, condition types and action types supported by event-condition-action approaches and (C) compensation techniques, scope, etc. for transactional approaches. However the study was time constrained and did not compare the application of these properties for different types of workflow approaches. Indeed in any such future study it may prove beneficial. It may be important to focus on how event types can be supported, how the state of a workflow can be defined, and how rules can be specified that transform the state of a workflow.
2. To further enhance understanding about common properties of sales order processing workflows. The domain analysis developed in this study is based on a number of simplifying assumptions. As a consequence the semi-generic set of sales order processing requirements established may prove limited in any specific application. The following limitations of the domain analysis were identified at the end of the study:
    - In the case of the logical failure of an activity it may not be necessary to re-execute all data-dependent activities because a planning stage will have previously been completed. For example assume that a sales order has been entered, the customer's creditworthiness has been checked and stock has been allocated. When the order is picked, assume that not all items are on stock. The current domain analysis indicates that in the case of such a logical failure the upstream activity and related downstream activities should be re-executed. This implies in the example case that the activities 'check creditworthiness' and 'allocate stock' should be redone. But this is not necessary anymore. Additional research might investigate how to overcome such problems.
    - Not necessarily all activities found on invalid paths of execution should be cancelled following a business rule change. For example, in specific cases it may be that the same activity is required along different paths of execution. The current domain analysis may therefore necessitate automatic triggering of actions that lead to inefficient or even unacceptable effects. Consider the

following specific example. Assume that the first activity in a workflow is 'maintain sales order' and that subsequently the workflow splits into two threads. Suppose also that the upper thread is triggered if the order quantity is larger than 100 so that goods are produced by a supplier. In this thread first an order confirmation is sent and next a purchase order is created. Suppose that the lower thread is triggered if the order quantity equals or is smaller than 100. In this lower thread suppose firstly stock is allocated, secondly an order confirmation is sent, and so on. Now assume that the upper thread has been triggered and the activity send order confirmation is performed. Suppose now the customer decides to order fewer products. Based on results of the current domain analysis an order cancellation is sent automatically. In practice this will not be acceptable to customers who have now ordered fewer products but have not cancelled the order, even though at some point in time the lower thread of execution will result in a new order confirmation to be send. Once again extended domain analysis could be carried out to further refine the workflow approach currently recommended in this thesis.

- In cases where an exception occurs it may prove inadequate to execute activities in a predefined order. Consider the case of a workflow that consists of a single thread of execution in which the first activity to be performed is 'maintain sales order properties', the second activity is 'allocate stock', and the third activity is 'check-creditworthiness'. This activity order might be designed to ensure that an order can be fulfilled before effort and time is spent checking the creditworthiness of a customer. Now assume that all three activities have been performed and that the customer wishes to increase the quantity ordered by 200. When using the developed approach these activities would need to be redone in the same order. However in practical situations of a high priority rush order it may be preferable for these activities to be re-performed in parallel. Once again further domain analysis could lead to refinement of workflow approach developed during this study.

3. To conduct a new domain analysis related to the kinds of exception that have relevance to workflow system developers which characterises their impact and necessary properties of exception handling structures. So far this study has only identified and developed properties of three general requirements. As yet it has provided little explanation as to the background of the properties of exceptions, nor of the characteristics of different types of exception handling specifications. Such a new domain analysis would require further study of (A) the states that any standard workflow may enter and the exceptions that may occur and (B) how different types of exception handling specification can be used to code up the handling of exceptions bearing in mind particular states and exception conditions.

4. To re-evaluate the extent to which the above enhancements to new workflow approach proposed and developed in this study can satisfy semi-generic sales order processor and workflow system developer requirements. Such an activity would naturally follow study under 1, 2 and 3 above, i.e. as the requirements of two different user groups are updated.

5. To propose alternative and better approaches to workflow specification and enactment when workflows are subject to uncertainty. For example it can be observed in chapter 7 that many rules have been defined, but as yet no rules have been defined to reverse parts of an exception workflow (albeit that such a possibility was mentioned as a concept in section 7.0 of chapter 7). Furthermore in the developed approach to creating exception workflows, their activity execution needs synchronisation once an exception workflow is made permanent. In some such cases it may prove better to immediately incorporate the effects in the standard workflow so the impact of all exceptions is made clear and inappropriate situations can be detected immediately. Yet another approach may be to combine the handling of some exceptions in groups in order to evaluate various combinations of similar and different exceptions, rather than treat every exception individually.

5.0 Future research

This research study has largely focussed on the application of workflow approaches within the domain of sales order processing. The scope of research needed to be constrained in order to carry out study in sufficient depth. However new understandings have been obtained within the study that may be applicable in other application domains or may offer a foundation for new research on semi-generic workflow systems.

5.1 Workflow simulation

Many of the assumptions upon which workflow simulation approaches previously reported in the literature have been based, are too simplistic to adequately facilitate exception handling in many practical situations. Firstly exception types, other than logical activity failures, may arise because of activity input changes and business rules changes (as discussed in chapter 5). In previous reported workflow simulation approaches [Aal01, Hee00, Des00, Bae99] only logical failure exception types are modelled. Secondly, previous literature assumes that if a task fails in its execution the task itself needs to be redone. But in the sales order processing domain at least it is likely that if a task fails other tasks will also need to be redone, not only the task that failed. Thirdly, previous simulation approaches do not take into account queue building as a consequence of the occurrence of activity batch-sizes or regular processing times (e.g. daily or weekly) and their effect on workflow performance indicators such as flow times, throughput times and resource utilisation. Instead it is assumed that items are processed on an item-by-item basis. But many such issues are of great relevance to the flow time of a workflow and in practice little time may remain for a workflow to complete when an exception occurs. Therefore it may be worthwhile to replace and evaluate some aspects of the mathematical definitions that underpin previous workflow simulation approaches by building (on the use of concepts and techniques) proposed in this thesis. Indeed there is a need for an extended workflow simulation approach that models relationships between (on the one hand) activities, batch size processing, activity dependencies, start times, exception handling priority, etc. and (on the other hand) workflow performance measures such as flow times, throughput times, and resource utilisation.

5.2 Distributed data management

In the 1980s many approaches to the management of distributed and replicated data were developed and described in the literature [Slo87, Tho90, Car91, Sch94, Cro96]. Yet to date few practical distributed application systems have been integrated effectively. An important issue in the management of distributed data is the way distributed copies of logical data items are kept up to date. In general either all copies are updated or none is modified [Dav84, Sto79]. But such an approach leads to performance, scalability and availability problems, particularly in cases where many items need to be frequently updated [Tha88, Sop91, Som97]. The proposed workflow approach offers a new understanding as to when and if various copies of data items need to be updated. For example the concept of a standard workflow can be deployed to explain that firstly data needs to be available at an application when the activity is initially performed. Potentially the concept of exception workflows offers a new way of capturing different versions of data items that must be kept.

5.3 Workflow modelling techniques.

The focus of previous workflow modelling research described in the literature has been on developing different approaches to workflow, i.e. traditional, event-condition-action and transactional workflow approaches. Capabilities of the different kind of approach need to be compared systematically, such as in the manner described in this thesis. This can help inform choices made, such as by industrialists wishing to use available technology. However additional fundamental research is needed to characterise more completely key aspects of the specification and enactment of workflows when exception conditions arise. Firstly research is needed into the definition of workflow state and formal rules required to transform workflow states. To date new approaches have been proposed without any formal basis of definition. Secondly further background research is needed to determine the advantages and disadvantages of explicitly specifying as opposed to dynamically generating, compensation activity. For example, will it prove most effective to use the former approach as an informal modelling tool and the latter approach to implement complex procedures? Thirdly the specification and enactment of workflows cannot be considered independently without associated consideration of requirements of end-users (e.g. sales order processor) and workflow system developers.

5.4 Workflow approaches in other application domains

Although domain analysis in this study (and associated requirement capture) has meant that the proposed workflow approach is specifically geared towards sales order processing applications, potentially the new concepts may have much broader applicability. For example it is highly probably that the proposed workflow approach can be deployed in areas with similar characteristic exception handling needs, such as in credit loan processing. In credit loan processing comparable activities (to sales order processing activities) will need to be performed such as 'enter request', 'check outstanding loans', 'determine conditions', 'confirm with customer' and 'receive payment'. In credit loan processing a number of steps may fail (e.g. a customer may have too many outstanding loans) or a customer may change the level of credit requested. Therefore arguably domain properties of the proposed approach might alternatively be effectively defined in a more abstract way, e.g. in terms of highly procedural workflow applications. It is also probably that the proposed workflow approach could be used in various other data flow intensive manufacturing applications, to provide a groupware overlay on software applications such as ERP and distributed product and process design applications, albeit that some concept and mechanism modifications may be necessary. Sales order processing can be viewed as being but one module of an ERP software system, possibly encompassing many other application modules designed to support purchase & works order processing and master production scheduling. Indeed from an exception handling standpoint purchase order processing has similarities with sales order processing. Often interrelationships coupling workflows processed by ERP modules are complex and currently their nature is not well understood.

The proposed approach also has relevance in domains involving distributed product and process design within the context of concurrent engineering. Indeed the author studied aspects of this domain immediately following his PhD study of the sales order processing domain. Typically in this class of CE domain hundreds of distributed design activities take place and complex flows of designs and changes to designs need to be managed. On the other hand, this application domain has many distinct requirements when compared with SOP applications. Consequently any new application of the proposed workflow concepts in the domain of distributed product and process design applications will require a retracing and reworking of many of the steps described in this thesis. Those main steps should be as follows.

1. Need to analyse the distributed product and process design domain in terms of the exceptions that may occur and their impact on workflows [Epp90, Kri93, Kri97a, Yas99]. Also to derive a set of requirements for workflow designers. The following kinds of question are likely to be raised. For example, can design exceptions (such as 'design flaws' and 'design maturation events') be represented by exception types supported by the new workflow approach, such as input change, logical failure, and business rule changes as discussed in chapter 5? Or will new exception types be required? Another key question which will be raised will concern the extent to which data and control flows can be modelled at the time of design or do key aspects of them only first become clear at run-time. Another important question that will be raised is whether the requirements of design engineers are similar to those of sales order processors.

2. To study the literature and identify various workflow approaches (and associated concepts and mechanisms) that can address some of the designer requirements. Recent CE research of the author identified 'traditional' approaches [AMI93, Ver96], 'design configuration management' approaches [Kit89, Kri95, Kri97b], and 'advanced transaction processing' approaches [Kim84, Ban85, Kla85, Fer90, Kai90, Nod90] for design applications as candidate sources of methods and techniques. A question that is likely to influence the choice of technology is whether compensation action can be usefully specified by a static description, or if dynamic generation is relevant or necessary.

3. To determine the capabilities and limitations of identified workflow approaches with reference to designer requirements.
4. To develop a new workflow approach that is capable of supporting both designer requirements and distributed product and process design application needs. An important question that remains to be answered here is whether the concept of standard and exception workflows (as proposed in this study) will apply in this distinctly different application domain.

References

[Aal97] W. M. P. van der Aalst, "Verification of workflow nets", *Lecture Notes in Computer Science*, Vol. 1248, 1997

[Aal98] W. M. P. van der Aalst, "The application of Petri nets to workflow management", *Journal of Circuits Systems and Computers*, Vol. 8, No. 1, 1998

[Aal99] W. M .P. van der Aalst, "Interorganisational workflows: An approach based on message sequence charts and Petri-Nets", *Systems-Analysis-Modelling-Simulation*, Vol. 34, No. 3, 1999, pp. 335-367

[Aal00a] W. van der Aalst, S. Jablonski, "Dealing with workflow change: identification of issues and solutions", *Computer Systems Science and Engineering*, Vol. 15, No. 5, 2000

[Aal00b] W. M. P. van der Aalst, "Workflow verification: finding control-flow errors using Petri-net-based techniques", *Lecture Notes in Computer Science*, Vol. 1806, 2000, pp. 161-183

[Aal00c] W. M. P. van der Aalst, A. H. M. ter Hofstede, "Verification of workflow task structures: a Petri-net-based approach", *Information Systems*, Vol. 25, No. 1, 2000, pp. 43-69

[Aal00d] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, J. Wainer, "Workflow modelling using proclets", in Proc. of the 7 Int. Conf. on Co-operative Information Systems, Eilat, 2000, pp. 198-209

[Aal01] W. M. P van der Aalst, "Re-engineering knock-out processes", *Decision Support Systems*, Vol. 30, No. 4, 2001

[Ago00] A. Agostini, G. De Michelis, "Improving flexibility of workflow management systems", *Lecture Notes in Computer Science*, Vol. 1806, 2000

[Alo96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Guenthoer, C. Mohan, "Advanced transaction models in workflow contexts", *Proceedings of the 1996 IEEE 12th International Conference on Data Engineering*, New Orleans, USA, 1996, pp. 574-581

[Alo97a] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, "Functionality and limitations of current workflow management systems", *IEEE Expert*, Vol. 12, No. 5, 1997, pp. 1-25

[Alo97b] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, "Functionalities and limitations of current workflow management systems", IBM Research Report, IBM Almaden Research Center, 1997

[AMI93] AMICE, CIMOSA: Open System Architecture for CIM, 2nd extended and revised version, Springer-Verlag, Berlin, 1993

[And00] B. Anderson, P. Dyson, "Reuse requires architecture" in L. Barroca, J. Hall, P. Hall, editors, *Software architectures, advances and applications*, Springer-Verlag, London, 2000

[Arc01] F. J. Arcelus, T. P. M. Pakkala, G. Srinivasan, "The temporary sale problem with unknown termination date", *INFOR Journal*, Vol. 39, No. 1, 2001, pp. 71 – 88

[Ash01] J. Ashayeri, W. J. Selen, "Order selection optimization in hybrid make-to-order and make-to-stock markets", *Journal of the Operational Research Society*, Vol. 52, No. 10, 2001, pp. 1098 – 1106

[Baa03] Baan, iBaan for Logistics, "http://www.baan.com/downloadlibrary/logistics_br.pdf", 2003

[Bae99] J. S. Bae, S. C. Jeong, Y. H. Seo, Y. H. Kim, S. K. Kang, "Integration of workflow management and simulation", *Computers and Industrial Engineering*, Vol. 37, No. 1-2, 1999

[Ban85] F. Bancilhon, W. Kim, H. F. Korth, "A model of CAD transactions", *Proceedings of the 11th international conference on very large databases*, Stockholm, Sweden, 1985, pp. 25 – 3

[Bar96] D. J. Barrett, L. A. Clarke, P. L. Tarr, A. E. Wise, "A framework for event-based software integration", *ACM Transactions on Software Engineering and Methodology*, Vol. 5, No. 4, October 1996, pp. 378-421

[Bar99a] L. Baresi, F. Casati, S. Castano, S. Ceri, M. G. Fugini, I. Mirbel, B. Pernici, G. Pozzi, P. Grefen, WIDE Workflow development methodology, ESPRIT project 20280, Politecnico di Milano, identifier 3027-6, May, 1999

[Bar99b] E. Bartezzaghi, R. Verganti, G. Zotteri, "Simulation framework for forecasting uncertain lumpy demand", *International Journal of Production Economics*, Vol. 59, No. 1, 1999, pp. 499 – 510

[Bar00] L. Barroca, J. Hall, P. Hall, *"Software architectures – advances and applications"*, Springer-Verlag, London. 2000.

[Bec00] J. Becker, M. Rosemann, C. von Uthmann, "Guidelines of business process modeling", *Lecture Notes in Computer Science*, Vol. 1806, 2000

[Ben84] I. Benbasat, "An analysis of research methodologies", in F. W. McFarlan editor *The information systems research challenge*, Harvard Business School Press, Boston, Massachusetts, 1984, pp. 47 – 85

[Ben87] I. Benbasat, D. K. Goldstein, M. Mead, "The case research strategy in studies of information systems", *MIS Quarterly*, Vol. 11, No. 3, 1987, pp. 369 - 386

[Ber94] J. I. Berman, H. H. Moore, J. R. Wright, "Classic and PROSE stories: enabling technologies for knowledge-based systems", *AT&T Technical Journal*, Vol. 73, No. 1, 1994, pp. 69 – 80

[Ber00] J. W. M. Bertrand, M. Zuijderwijk, H. M. H. Hegge, "Using hierarchical pseudo bills of material for customer order acceptance and optimal material replenishment in assemble to order manufacturing of non-modular products", *International Journal of Production Economics*, Vol. 66, No. 2, 2000, pp. 171 – 184

[Bon85] T. V. Bonoma, "Case research in marketing: opportunities, problems, and a process", *Journal of Marketing Research*, Vol. 22, No. 2, 1985, pp. 199 - 208

[Bos99] J. Bosch, P. Molin, "Software architecture design: evaluation and transformation", *Proceedings of IEEE Symposium of Engineering of Computer Based Systems*, ?, April, 1999

[Bra95] P. Bradley, J. Browne, S. Jackson, H. Jagdev, "Business process reengineering - a study of the software tools currently available", *Computers in Industry*, Vol. 25, No. 3, 1995

[Bre93] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, G. Weikum, "Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows", *ACM Special Interest Group on Management of Data (SIGMOD) Record*, Vol. 22, No. 3, 1993, pp. 23-30

[Bro96] J. Browne, J. Harhen, J. Shivnan, *Production management systems,* Addison-Wesley Publishing Company Inc., Harlow, England, 1996

[Buc86] A. P. Buchmann, R. S. Carrera, M. A. Vazquez-Galindo, "Generalized constraint and exception handler for an object-oriented CAD-DBMS", *Proceedings of the 1986 international workshop on object-oriented database*, Pacific Grove, CA, USA, 1986

[Buc88] A. P. Buchmann, U. Dayal, "Constraint and exception handling for design, reliability and maintainability", *Managing engineering data: emerging issues*, San Francisco, CA, USA, 1998

[Bur95] R. Burman, *Manufacturing management :principles and systems*, McGraw-Hill, London, England, 1995

[Bus96] Bussler, "Specifying enterprise processes with workflow modeling languages", *Concurrent Engineering – Research and Applications*, Vol. 4, No. 3, 1996

[Car91] M. J. Carey, M. Livny, "Conflict detection tradeoffs for replicated data", *ACM Transactions on Database Systems*, Vol. 16, No. 4, 1991, pp.703-746

[Cas96] F. Casati, S. Ceri, B. Pernici, G. Pozzi, "Deriving active rules for workflow enactment", *Lecture Notes in Computer Science*, Vol. 1134, 1996

[Cas97] N. Cassaigne, M. Kroemker, M. G. Singh, S. Wurst, "Decision support for effective bidding in a competitive business environment", *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, 1997, pp. 3591 - 3596

[Cas98] N. Cassaigne, M. G. Singh, "Decision support for the pricing of services in business to business sale", *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Vol. 5, 1998, pp. 4752 – 4757

[Cas99] F. Casati, S. Ceri, S. Paraboschi, G. Pozzi, "Specification and implementation of exceptions in workflow management systems", *ACM Transactions on Database Systems*, Vol. 24, No. 3, 1999

[Chi99] D. K. W. Chiu, Q. Li, K. Karlapalem, "A meta modeling approach to workflow management systems supporting exception handling", *Information Systems*, Vol. 24, No. 2, 1999

[Cho00] H. R. Choi, H. S. Kim, Y. J. Park, K. H. Kim, M. H. Joo, H. S. Sohn, "Sales agent for part manufacturers: VMSA", *Decision Support Systems*, Vol. 28, No. 4, 2000, pp. 333 – 346

[Coc82] N. Cocco, S. Dulli, "Mechanism for exception handling and its verification rules", *Computer Languages*, Vol. 7, No. 2, 1982, pp. 89-102

[Cox89] I. J. Cox, N. H. Gehani, "Exception handling in robotics", *Computer*, Vol. 22, No. 3, 1989, pp. 43-49

[Cro96] J. Crowcroft, *Open Distributed Systems*, UCL Press, London, 1996

[Cug98] G. Cugola, "Tolerating deviations in process support systems via flexible enactment of process models", *IEEE Transaction on Software Engineering*, Vol. 24, No. 11, 1998

[Dav84] S. B. Davidson, "Optimism and consistency in partitioned distributed database systems", *ACM Transactions on Database Systems*, Vol. 9, No. 3, 1984, pp. 456-481

[Dell00] C. Dellarocas, M. Klein, "A knowledge-based approach for designing robust business processes", *Lecture Notes in Computer Science*, Vol. 1806, 2000.

[Der96] G. Derszteler, "Workflow management cycle - An integrated approach to the modelling, execution, and monitoring of workflow-based processes", *Wirtschaftsinformatik*, Vol. 38, No. 6, 1996

[Des00] J. Desel, T. Erwin, "Modeling, simulation and analysis of business processes", *Lecture Notes in Computer Science*, Vol. 1806, 2000

[Dil98] K. A. Dilger, "Design by desire", *Manufacturing Systems*, Vol. 16, No. 3, 1998, pp. 62-66, 68, 70, 73

[Dil01] K. A. Dilger, "Fit to order", *Manufacturing Systems*, Vol. 19, No. 4, 2001, pp. 57 – 62

[Dog00] A. Dorgac, Y. Tambay, A. Tumer, M. Ezbiderli, N. Tarbul, N. Hamali, C. Icdem, C. Beeri, *A workflow system through cooperating agents*, technical report, METU, March, 2000, http://www.srdc.metu.edu.tr/publications.html.

[Dou95] J. R. Dougherty, "Master planning: a key to satisfying customers in the '90s", *Proceedings of the 38th APICA International Conference and Exhibition*, APICS, Orlando, USA, 1995, pp. 80 – 84

[Dre94] S. J. Drew, K. J. Gough, "Exception handling: expecting the unexpected", *Computer Languages*, Vol. 20, No. 2, 1994, pp. 69-87

[Du97] W. Du, A. Elmagarmid, "Workflow management: state of the art vs. state of the products", *HP Laboratories Technical Report*, No.97-90, 1997, pp. 1-21

[Ede97] J. Eder, W. Liebhart, "Workflow transactions", in P. Lawrence editor *Workflow Handbook,* Wiley & Sons, 1997, pp. 195-202

[Ell95] C.A Ellis, K. Keddara, G. Rozenberg, "Dynamic change within workflow systems". *Proceedings of Conference on Organisational Computing Systems*, New York, USA, 1995, pp. 10-21

[Ell00] S. Ellis, K. Keddara, "A workflow change is a workflow", *Lecture Notes in Computer Science*, Vol. 1806, 2000, pp. 201-217

[Epp90] S. Eppinger, D. Whitney, R. Smith, D. Gebala, "Organising the tasks in complex design projects", *Proceedings of the ASME Design Theory and Methodology Conference*, Chicago, 1990, pp. 39-46

[Fer90] M. F. Fernandez, S. B. Zdonik, "Transaction groups: a model for controlling co-operative transactions", *Proceedings of the third international conference on persistent object systems*, Newcastle, Australia, 1990, pp. 341 – 352

[Fit96] B. Fitzgerald, C. Murphy, "Business process reengineering: putting theory into practice", Infor, Vol. 34, No.1, 1996

[For02] C. Forza, F. Salvador, "Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems", *International Journal of Production Economics*, Vol. 76, No. 1, 2002, pp. 87 – 98

[Gan95] S. Ganesan, E. D. Robe, D. A. Roth, E. C. Chung, C. A. Vassiliadis, "Intelligent order entry system for portable industrial air compressors in a manufacturing environment", *Expert Systems with Applications*, Vol. 9, No. 2, 1995, pp. 223 – 236

[Gar87a] H. Garcia-Molina, R. K. Abbott, "Reliable distributed database management", *Proceedings of the IEEE*, Vol. 75, No. 5, 1987, pp. 601-620

[Gar87b] H. Garcia-Molina, K. Salem, "Sagas", *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data*, San Francisco, California, USA, 1987, pp. 249-259

[Gar91] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, K. Salem, "Modeling long-running activities as nested Sagas", *Data Engineering Bulletin*, Vol. 14, No. 1, 1991, pp. 14-18

[Geo95] D. Georgakopoulos, M. Hornick, A. Sheth, "An overview of workflow management – from process modeling to workflow automation infrastructure", *Distributed and Parallel Databases*, Vol. 3, No. 2, 1995

[Gil00] M. Gillmann, G. Weissenfels, G. Weikum, A. Kraiss, "Performance and availability assessment for the configuration of distributed workflow management systems", *International Conference on Extending Database Technology*, Konstanz, Germany, 2000

[Gre00] P. Green, M. Rosemann, "Integrated process modeling: an ontological evaluation", *Information Systems*, Vol. 25, No. 2, 2000

[Hag00] C. Hagen, G. Alonso, "Exception handling in workflow management systems", *IEEE Transactions on Software Engineering*, Vol. 26, No. 10, 2000

[Ham93] M. Hammer, J. Champy, *Reengineering the corporation: a manifesto for business revolution*, Harper Business, New York, 1993

[Har95] B. Harris, H. Frederick, J. P. Pinder, "Revenue management approach to demand management and order booking in assemble-to-order manufacturing", *Journal of Operations Management*, Vol. 13, No. 4, 1995, pp. 299 – 309

[Haz86] N. Hazeltine, "From prototype to production: the maturing of ocean's user interface", *Proceedings of the First Human-Computer Interaction U.S. A. -Japan Conference*, 1986

[Hee00] K. M. van Hee, H. A. Reijers, "Using formal analysis techniques in business process redesign", in editors W. M. P. van der Aalst, J. Desel, A. Oberweis, *Lecture Notes in Computer Science,* Vol. 1806, Springer-Verlag, 2000, pp. 142-160

[Hei93] J. Heizer, B. Render, *Production and operations management: strategies and tactics,* Prentice Hall Inc., Englewood Cliffs, New Jersey, USA, 1993

[Hir95] B. E. Hirsch, M. Kroemker, K. -D. Thoben, A. Wickner, "BIDPREP - an approach for simultaneous bid preparation", *Computers in Industry*, Vol. 26, No. 3, Aug 1995, pp. 273 – 279

[Ho02] T. H. Ho, S. Savin, C. Terwiesch, "Managing demand and sales dynamics in new product diffusion under supply constraint", *Management Science*, Vol. 48, No. 2, 2002, pp. 187 – 206

[Hol97] A. W. Holt, *Organised Activity and its Support by Computer,* Kluwer Academic Pub., Dordrecht, Holland, 1997

[ICA81] -, *Integrated computer aided manufacturing architecture Part II, Volume IV - functional modeling manual (IDEF0)*, Air Force Materials Laboratory, Wright-Patterson Air Force Base, Ohio 45433, AFWAL-TR-81-4023, 1981

[Im99] I. Im, O. A. El Sawy, A. Hars, "Competence and impact of tools for BPR", *Information & Management*, Vol. 36, No. 6, 1999

[Jab00] S. Jablonski, "Workflow management between formal theory and pragmatic approaches", *Lecture Notes in Computer Science*, Vol. 1806, 2000

[Jan00] G. K. Janssens, J. Verelst, B Weyn. "Techniques for modelling workflows and their support of reuse", in W. M. P. van der Aalst, J. Desel, A. Oberweis, editors, *Business Process Management-Models, Techniques and Empirical Studies*, Springer-Verlag, Berlin, Heidelberg, 2000, pp. 1-15

[Joo97] S. M. M. Joosten, M. Schipper, *Improving your business – think processes*, Anaxogoras, 1997

[Kai90] G. E. Kaiser, "A flexible transaction model for software engineering", *Proceedings of the sixth international conference on data engineering*, Los Angeles, California, USA, 1990, pp. 560 - 567

[Kal99] J. Kallio, T. Saarinen, S. Salo, M. Tinnila, A. P. J. Vepsalainen, "Drivers and tracers of business process changes", *Journal of Strategic Information Systems*, Vol. 8, No. 2, 1999

[Kam95] M. Kamath, K. Ramamritham, “Modeling, correctness & systems issues in supporting advanced database applications using workflow management systems”, *University of Massachusetts Computer Science Technical Report*, No. 95-50, 1995

[Kam96] M. Kamath, K. Ramamritham, "Correctness issues in workflow management", *Distributed Systems Engineering (DSE) Journal : Special issue on workflow management systems*, Vol. 3, No. 4, 1996

[Kam98] M. Kamath, K. Ramamritham, "Failure handling and coordinated execution of concurrent workflows", *Proceedings of 14th International Conference on Data Engineering (ICDE'98)*, Orlando, Florida, 1998

[Kat93] D. Katz, S. Manivannan, “Exception management on a shop floor using online simulation”, *Proceedings of the winter simulation conference*, Los Angeles, CA, USA, 1993, pp. 888-896

[Kay96] A. McKay, F. Erens, M. S. Bloor, “Relating product definition and product variety”, *Research in Engineering Design*, Vol. 8, No. 2, 1996, pp. 63 – 80

[Ket97] W. J. Kettinger, J. T. C. Teng, S. Guha, "Business process change: a study of methodologies, techniques, and tools", *MIS Quarterly*, Vol. 21, No. 1, 1997

[Kim84] W. Kim, R. A. Lorie, D. McNabb, W. Plouffe, “A transaction mechanism for engineering design databases”, *Proceedings of the 10th international conference on very large databases*, Singapore, 1984, pp. 355 –362

[Kim96] C. Kim, "A comprehensive methodology for business process reengineering", *Journal of Computer Information Systems*, Vol. 37, No. 1, 1996

[Kin93a] B. Kingsman, L. Worden, L. Hendry, A. Mercer, E. Wilson, “Integrating marketing and production planning in make-to-order companies”, *International Journal of Production Economics*, Vol. 30-31, 1993, pp. 53 – 66

[Kin93b] B. Kingsman, L. Hendry, E. Wilson, “Decision support system for the dynamic planning of customer order intake and capacity resources for make-to-order companies”, *IFIP Transactions B: Computer Applications in Technology*, No. B-13, 1993, pp. 133 – 141

[Kin96] B. Kingsman, L. Hendry, A. Mercer, A. de Souza, “Responding to customer enquiries in make-to-order companies: Problems and solutions”, *International Journal of Production Economics*, Vol. 46-47, 1996, pp. 219 – 231

[Kin00] E. Kindler, A. Martens, W. Reisig, “Interoperability of workflow applications”, in W. M. P. van der Aalst, J. Desel, A. Oberweis, editors, *Business Process Management-Models, Techniques and Empirical Studies*, Springer-Verlag, Berlin, Heidelberg, 2000, pp. 235-253

[Kit89] H. Kitagawa, N. Ohbo, “Design data modeling with versioned conceptual configuration”, *Proceedings of the Thirteenth Annual International Computer Software & Applications Conference - COMPSAC*, Orlando, USA, September 20-22, 1989, pp. 225-233

[Kla85] P. Klahold, G. Schlageter, R. Unland, W. Wilkes, "A transaction model supporting complex applications in integrated information systems", *Proceedings of the international conference on management of data*, Austin, Texas, USA, 1985, pp. 388 – 401

[Kle00] M. Klein, C. Dellarocas, Bernstein, "Special issue on adaptive workflow systems", *Computer-Supported Collaborative Work*, Vol. ?, 2000

[Kle95] M. Klein, "Conflict management as part of an integrated exception handling approach", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 9, No. 4, 1995, pp. 259-267

[Kno98] G. F. Knolmayer. "Business rules layers between process and workflow modelling an object oriented perspective", in S. Demeyer, J. Bosch, *Object Oriented Technology*, Springer, 1998, pp. 205-297

[Knu87] J. L. Knudsen, "Better exception-handling in block-structured systems", *IEEE Software*, Vol. 4, No. 3, 1987, pp. 40-49

[Kra98] L. J. Krajewski, L. P. Ritzman, *Operations management: strategy and analysis*, Addison-Wesley Publishing Company Inc., Reading, Massachusetts, USA, 1998

[Kri93] V. Krishnan, S. D. Eppinger, D. E. Whitney, "Iterative overlapping: accelerating product development by preliminary information exchange", *Proceedings of the ASME Design Theory and Methodology Conference*, DE-vol 53, 1993, pp. 223-231

[Kri95] K. Krishnamurthy, K. H. Law, "Configuration management in a CAD paradigm", *Proceedings of the 1995 ASME International Mechanical Engineering Congress and Exposition*, San Francisco, USA, November 12-17, 1995, pp. 103-116

[Kri97a] K. Krishnamurthy, K. H. Law, "Data management model for collaborative design in a CAD environment", *Engineering with Computers*, Vol. 13, No. 2, 1997, pp. 65-86

[Kri97b] K. Krishnamurthy, K. H. Law, "Data management model for collaborative design in a CAD environment", *Engineering with Computers*, Vol. 13, No. 2, 1997, pp. 65-86

[Kue98] P. Kueng, "Impact of workflow systems on people, tasks and structure", *Proceedings of the 5th European Conference on the Evaluation of IT.*, Reading, UK, Nov., 1998, pp. 67-75

[Kuo99] R. J. Kuo, K. C. Xue, "Fuzzy neural networks with application to sales forecasting", *Fuzzy Sets and Systems*, Vol. 108, No. 2, 1999, pp. 123 – 143

[Lee98] R. M. Lee, *Candid – a formal language for electronic contracting*, Euridis Research Monograph (RM 1988.08.02), 1998

[Ley95] F. Leymann, "Supporting business transactions via partial backward recovery in workflow management systems", *Proceedings of 6th German Conference on Database Systems in OAEce, Engineering and Scientific Applications*, Vol. 6, Dresden, Germany, 1995, pp. 51-70

[Lis79] B. H. Liskov, A. Snyder, "Exception handling in CLU", *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 6, 1979, pp. 546-558

[Luc95a] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, W. Mann, "Specification and analysis of system architecture using Rapide", *IEEE Transactions on Software Engineering*, Vol. 21, No. 4, April, 1995, pp. 336-355

[Luc95b] D. C. Luckham, J. Vera, "An event-based architecture definition language", *IEEE Transactions on Software Engineering*, Vol. 21, No. 9, September, 1995, pp. 717-734

[Lus93] M. Luscombe, *MRPII: Integrating the business,* Butterworth-Heinemann Ltd., Oxford, England, 1993

[Mar88] D. A. Marca, C. L. McGowan, *Structured analysis and design technique*, McGraw-Hill, New York, 1988

[Mas90] M. Hasegewa, M. Takata, T. Temmyo, H. Hatsuka, "Modelling of exception handling in manufacturing cell control and its application to PLC programming", *Proceedings of the 1990th IEEE internal conference on robotics and automation*, Cincinnati, OK, USA, 1990

[May92] R. J. Mayer, T. P. Cullilane, P. S. de Witte, W. B. Knappenberger, B. Perakath, M. S. Wells, *Information integration for concurrent engineering IDEF3 process description capture method report*, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433, AL-TR-1992-0057, 1992

[Mer95] K. Mertins, H. Edeler, R. Jochem, J. Hofmann, "Object-oriented modelling and analysis of business processes", in P. Ladet, F. B. Vernadat, editors, *Integrated Manufacturing Systems Engineering*, Chapman & Hall, London, UK, 1995, pp. 115-128

[Mey91] G. R. Meijer, L. O. Hertzberger, T. L. Mai, E. Gaussens, F. Arlabosse, "Exception handling system for autonomous robots based on PES", *Robotics and Autonomous Systems*, Vol. 7, No. 2-3, 1991, pp. 197-209

[Mey00] M. Meyer, "On the practical relevance of an integrated workflow management system", in W.M.P. van der Aalst, J. Desel, A. Oberweis, editors, *Business Process Management-Models, Techniques and Empirical Studies*, Springer-Verlag, Berlin, Heidelberg, 2000, pp. 315-327

[Mil58] G. Mills, O. Standingford, *Office organization and method,* Second Edition, Sir Isaac Pitman & Sons Ltd., London, 1958

[Moh95] C. Mohan, G. Alonso, R. Guenthoer, M. Kamath, "Exotica: a research perspective on workflow management systems", *Data Engineering Bulletin (Special Issue on Infrastructure for Business Process Management)*, Vol. 18, No. 1, pp. 19-26, 1995

[Mue00] R. Mueller, E. Radim, "Dealing with logical failures for collaborating workflows", *Proceedings of the 7th International Conference on Coop IS*, Eilat, Sept., 2000, pp. 210-222

[Mon87] A. R. Montazemi, D. W. Conrath, C. A. Higgens, "Exception reporting information system for ill-structured decision problems", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-17, No. 5, 1987, pp. 771-779

[Mur61] A. A. Murdoch, J. R. Dale, *The clerical function,* Sir Isaac Pitman & Sons Ltd., London, 1961

[Nav03] Navision, Axapta Logistics, http://www.navision.com/hq/mediafiles/newdoc/PDF_AXAPTA/6/-Logistics_ax30.pdf, 2003

[Nei99] P. O'Neill, A. S. Sohal, "Business process reengineering - a review of recent literature", *Technovation*, Vol. 19, No. 9, 1999

[Nod90] M. H. Nodine, S. B. Zdonik, "Co-operative transaction hierarchies: a transaction model to support design applications", *Proceedings of the 16th international conference on very large databases*, Brisbane, Australia, 1990, pp. 83 – 94

[Obe91] A. Oberweis, W. Stucky, "Exception handling in software systems: a literature survey", *Wirtschaftsinformatik*, Vol. 33, No. 6, 1991, pp. 492-502

[Olh01] J. Olhager, M. Rudberg, J. Wikner, "Long-term capacity management: Linking the perspectives from manufacturing strategy and sales and operations planning", *International Journal of Production Economics*, Vol. 69, No. 2, 2001, pp. 215 – 225

[Orf94] R. Orfali, D. Harkey, J. Edwards, *Essential Client/Server Survival Guide*, John Wiley & Sons, Inc., New York, 1994

[Orf97] R. Orfali, D. Harkey, *The client/server programming with JAVA and CORBA*, John Wiley & Sons Inc., New York, 1997

[Pap99] M. P. Papazoglou, W. J. van den Heuvel, "Leveraging legacy assets", in M. P. Papazolou, S. Spaccapletra, Z. Tari, editors, *Advances in Object Oriented Modelling*, New York, MIT Press, 1999

[Par02] K. Parker, "Events happen, but demand is always", *Manufacturing Systems*, Vol. 20, No. 2, 2002, pp. 40 – 43

[Pun84] L. Pun, *Systèmes industriels d'intelligence artificielle; Outils de productique*, Editests, Paris (in French), 1984

[Reu95] A. Reuter, F. Schwenkreis, "ConTracts: a low-level mechanism for building general-purpose workflow management systems", *Bulletin of the Technical Committee on Data Engineering*, Vol. 18, No. 1, 1995

[Ric87] J. A. Ricketts, R. R. Nelson, "Management-by-exception reporting: an empirical investigation", *Information & Management*, Vol. 12, No. 5, 1987, pp. 235-246

[Ros77a] D. T. Ross, "Structured analysis: a language for communicating ideas", *IEEE Transactions on software engineering*, 1977, SE-3, pp. 16-24

[Ros77b] D. T. Ross, K. E. Schoman, "Structured analysis for requirements definition", *IEEE Transactions on software engineering*, 1977, SE-3, pp. 6-15

[Sad00a] W. Sadiq, M. E. Orlowska, "Analyzing process models using graph reduction techniques", *Information Systems*, Vol. 25, No. 2, 2000, pp. 117-134

[Sad00b] S. W. Sadiq, O. Marjanovic, M. E. Orlowska, "Managing change and time in dynamic workflow processes", *International Journal of Cooperative Information Systems*, Vol. 9, No. 1-2, 2000

[Sch93] F. Schwenkreis, "APRICOTS - a prototype implementation of a ConTract system - management of the control flow and the communication system", *Proceedings of the 12th Symposium on Reliable Distributed Systems*, Princeton, NJ, USA, 1993, pp.12-21

[Sch94] G. Schussel, "Database replication: playing both ends against the middleware", *Client/server Today*, November, 1994, pp. 57-67

[Sch96] T. Schäl, "Workflow management systems for process organisations", *Lecture Notes in Computer Science,* Vol. 1096, 1996, pp. 44-49

[Sch98] A. W. Scheer, *ARIS – Business process frameworks*, Second Edition, Berlin, 1998

[Sch00] A. W. Scheer, M. Nuttgens, "ARIS architecture and reference models for business process management", *Lecture Notes in Computer Science*, Vol. 1806, 2000

[Sha93] S. M. Sharman, "CGS - the ICL configurer graphics service", *ICL Technical Journal*, Vol. 8, No. 4, 1993, pp. 589 – 602

[She96] A. Sheth, D. Georgakopoulos, S. M. M. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, A. Wolf, "Report from the NSF workshop on workflow and process automation in information systems", *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions*, Georgia, USA, May 8-10, 1996

[She99] A. Sheth, "Workflow process management and enterprise application integration in healthcare", *IEEE Workshop on enterprise networking and computing in healthcare industry*, Sydney, Australia, 1999

[Sim94] G. C. Simsion, "A methodology for business process reengineering", *IFIP Transactions A - Computer Science and Technology*, Vol. 54, 1994

[Slo87] M. Sloman, J. Kramer, *Distributed Systems and Computer Networks*, Prentice-Hall International, London, 1987

[Som97] A. K. Somani, N. H. Vaidya, "Understanding fault tolerance and reliability", *Computer*, Vol. 30, No. 4, 1997, pp. 45-50

[Sop91] N. Soparkar, H. F. Korth, A. Silberschatz, "Failure-resilient transaction management in multidatabases", *Computer*, Vol. 24, No. 12, 1991, pp. 28-36

[Spu96] G. Spur, K. Mertins, R. Jochem, *Integrated enterprise modelling*, Beuth Verlag, Berlin

[Sti90] R. S. Stieber, A. M. Agogino, J. Sullivan, "Expert advisor for configuring information display systems", *Proceedings of the 1990 ASME International Computers in Engineering*, Boston, MA, USA, 1990

[Sto78] E. Stone, *Research methods in organisational behavior*, Scott, Foresman and Company, Glenview, Illinois, 1978

[Sto79] M. Stonebraker, "Concurrency control and consistency of multiple copies of data in distributed Ingres", *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 3, 1979, pp. 188-194

[Sym69] M. Symes, *Office procedures and management*, Heinemann Ltd., London, 1969

[Tid00] G. Tidhar, E. A. Sonenberg, *Organised distributed systems*, Technical Report, University of Melbourne, Australia, June, 2000

[Tha88] C. Thanos, E. Bertino, C. Carlesi, "The effects of two-phase locking on the performance of a distributed database management system", *Performance Evaluation*, Vol. 8, No. 2, 1988, pp. 129-157

[Tho90] G. Thomas, G. R. Thompson, C.-W. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, B. Hartman, "Heterogeneous distributed database systems for production use", *ACM Computing Surveys*, Vol. 22, No. 3, 1990, pp. 237-266

[Tra00] G. Trajcevski, C. Baral, J. Lobo, "Formalising (and reasoning about) the specification of workflows", *Proceedings of the 7th International Conference* on *Coop IS*, Eilat, Sept., 2000, pp. 1-17

[Tru95] A. Truksa, "BMW rolls out its showroom assistants", *Siemens Review*, Vol. 62, No. 3-4, 1995, pp. 11

[Uma97] A. Umar, *Object-oriented client/server internet environments*, Prentice Hall Ltd., London, 1997

[Val98] R. Valk, "Petri-Nets as token objects. An introduction to elementary object nets". *Proceedings of the 19th International Conference on Applications and Theory of Petri-nets*, ?, 1998

[Ver96] F. B. Vernadat, *Enterprise modeling and integration: principles and applications*, Chapman & Hall, London, UK, 1996

[Vid94] R. Vidgen, J. Rose, B. Wood, T. Woodharper, "Business process reengineering - the need for a methodology to revision the organisation", *IFIP Transactions A - Computer Science and Technology*, Vol. 54, 1994

[Voo00] M. Voorhoeve, "Compositional modeling and verification of workflow processes", *Lecture Notes in Computer Science*, Vol. 1806, 2000, pp. 184-200

[Wae92] H. Waechter, A. Reuter, "The ConTract Model", in A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 7, pp. 219-263, Morgan Kaufmann Publishers, San Mateo, 1992

[WFM94] D. Hollingsworth, *The workflow reference model*, Document Number TC00-1003, v1.1, Workflow Management Coalition, Brussels, Belgium, 1994, URL: http://www.aiim.org/wfmc/

[Wid96] J. Widom, S. Ceri, *Active database systems: triggers and rules for advanced database processing*, Morgan Kaufmann Publishers, San Francisco, USA, 1996

[Yas99] A. A. Yassine, K. R. Chelst, D. R. Falkenburg, "A decision analytic framework for evaluating concurrent engineering", *IEEE Transactions on Engineering Management*, Vol. 46, No. 2, 1999, pp. 144-157

[Yin84] R. K. Yin, *Case study research, design and methods*, Sage Publications, Beverly Hills, California, 1984

[Zap00] M Zapf, A. Heinzl, "Evaluation of genuine process design patterns", in W. M. P. van der Aalst, J. Desel, A. Oberweis, editors, *Business Process Management-Models, Techniques and Empirical Studies*, Springer-Verlag, Berlin, Heidelberg, 2000, pp. 82-98