# Appendix A:

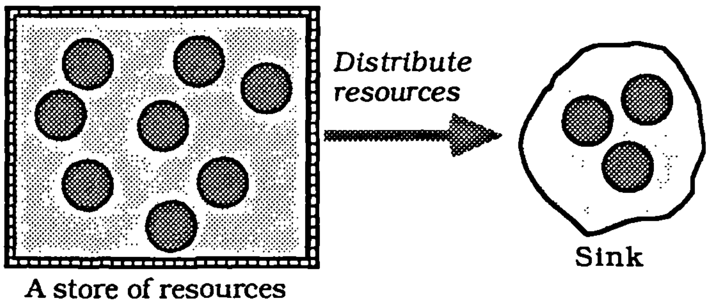
## Set of Identified Domain Abstractions

Vol II

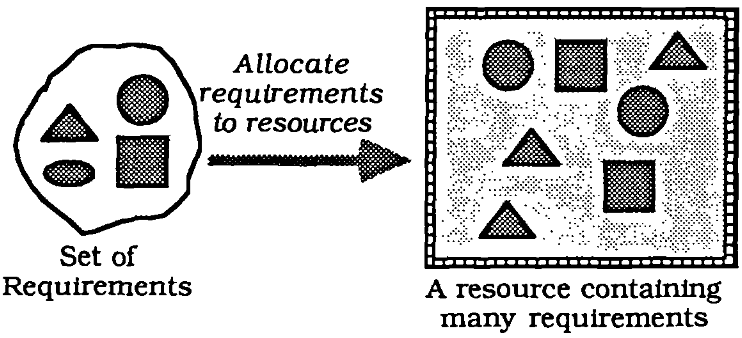
02023808<del>4</del>

## 1: Basic Object Containment and Object Allocation Domains

Many abstract domain models are developed from the two basic models of object containment and object allocation, as demonstrated in the remainder of this Appendix. Object containment domains have a store of objects which are gradually depleted as they are removed to a store, as shown below:

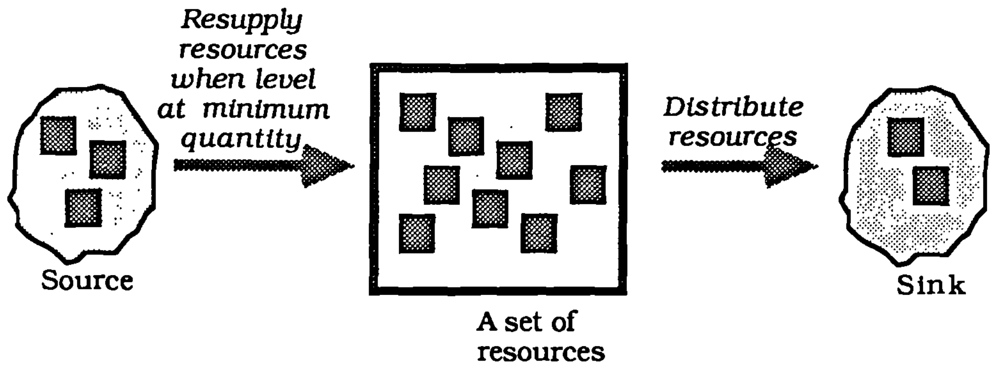



Object Allocation is more complex and involves the allocation of requirements to a resource if some constraints are met, constraints symbolised by the different shapes of the requirements.

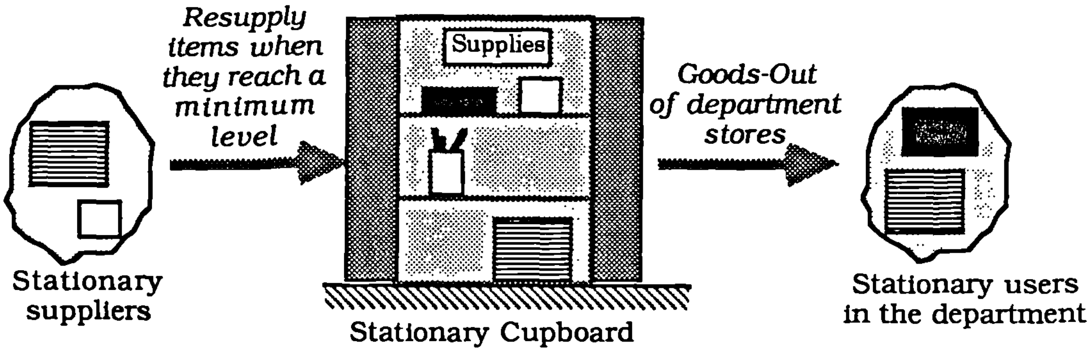


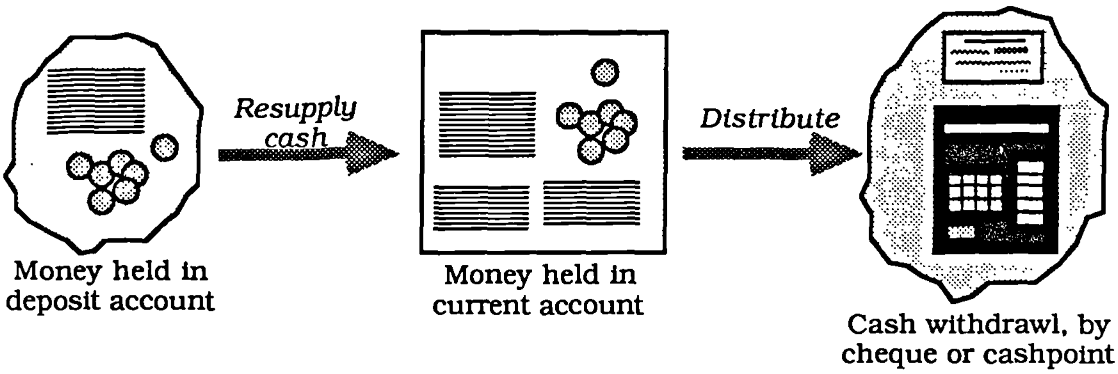
## 2: Basic Renewable Resource Management

A set of resources are held in a store which controls the level of resource held in the store. Resources are depleted over time and periodically replaced by new resources provided by a source when the level of resources reaches a minimum quantity.



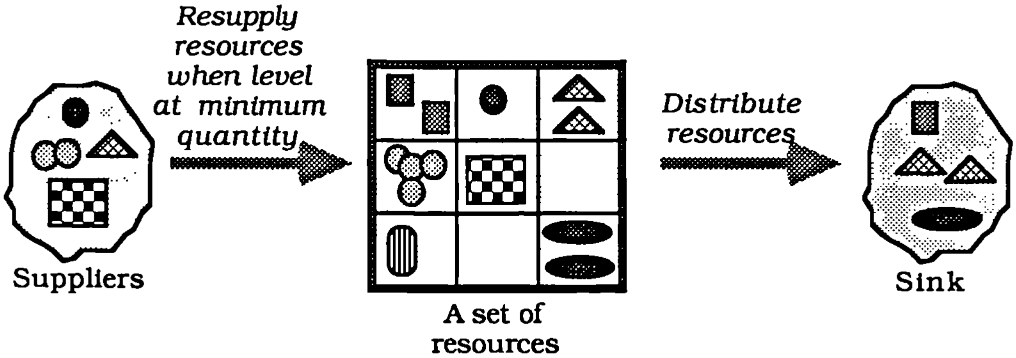

Example: This abstraction supports an analogy between domains for controlling the level of stationary supplies in a university department's cupboard and for transferring money from deposit to current accounts for all relevant customers of a local bank on a daily basis. In both resupply occurs when the level of stationary in the cupboard and a minimum level of money in the current account are reached.



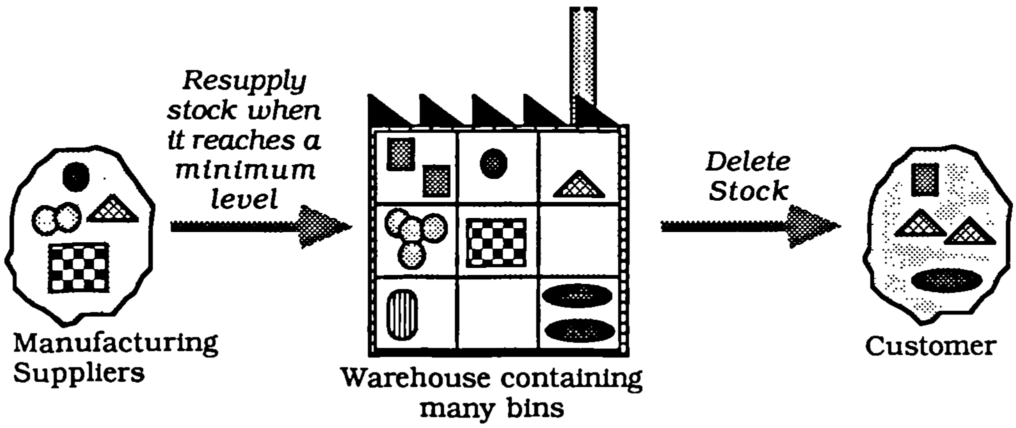



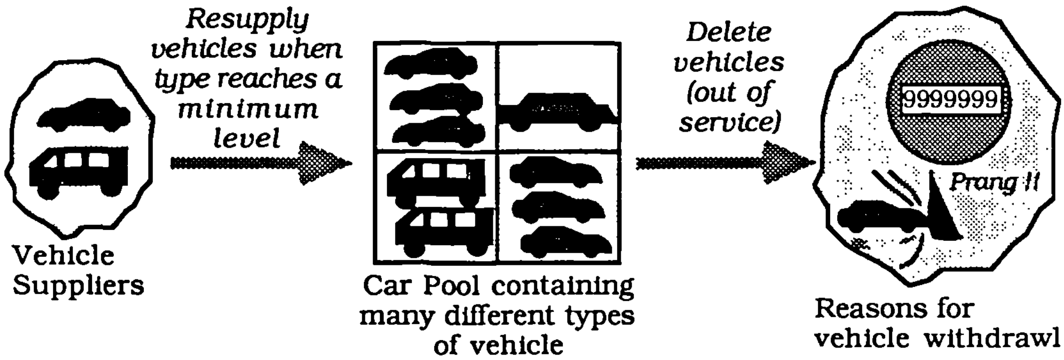
### 3: Structured Renewable Resource Management

A set of resources are held in a store which is structured and divided into segments acting as mini-stores controlling the level of resource type held in each. Resources are depleted over time and periodically replaced by new resources provided by a supplier when the level of resources in a mini-store reaches a minimum quantity.



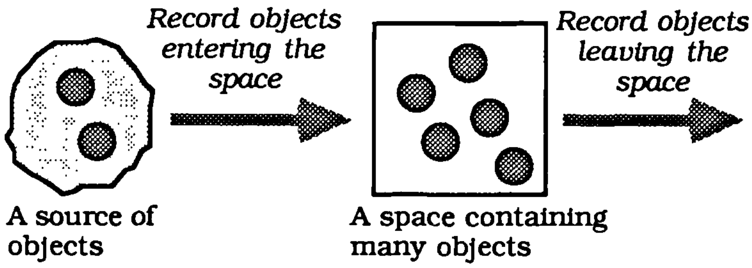

Example: This abstraction supports an analogy between domains for stock control in warehouses and maintaining the fleet of cars in a large car rental firm. Both involve the gradual depletion of a stock (through use, accidents etc.) and the renewing of stock segements when predetermined limits are reached.



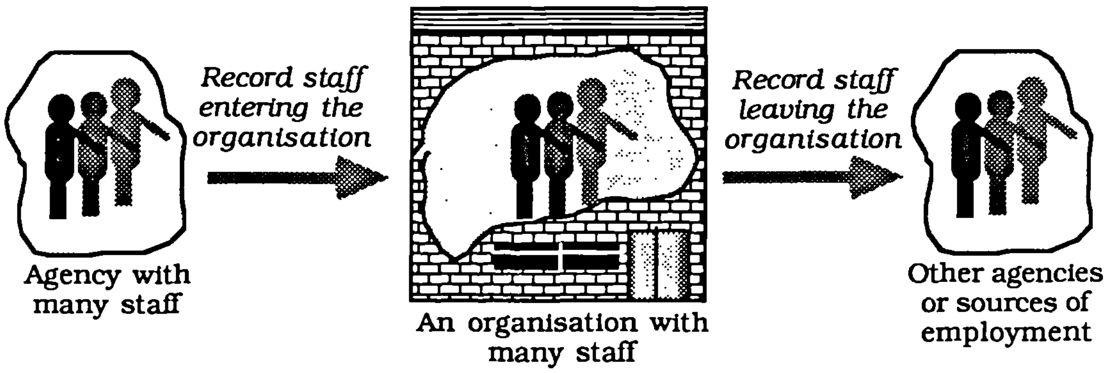



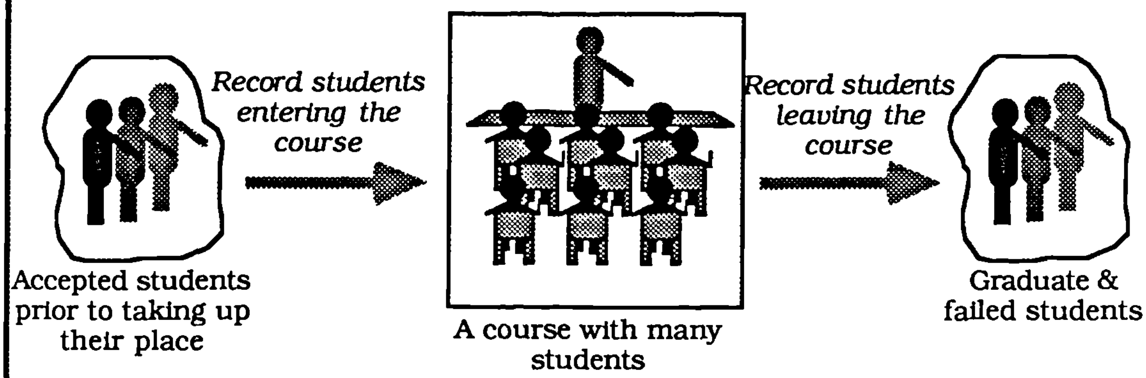
## 4: Object Recording

Objects move in and then out of a space. The aim of the information system is to record that movement to provide information on objects which are in the space. It represents one of the simplest domain abstractions.



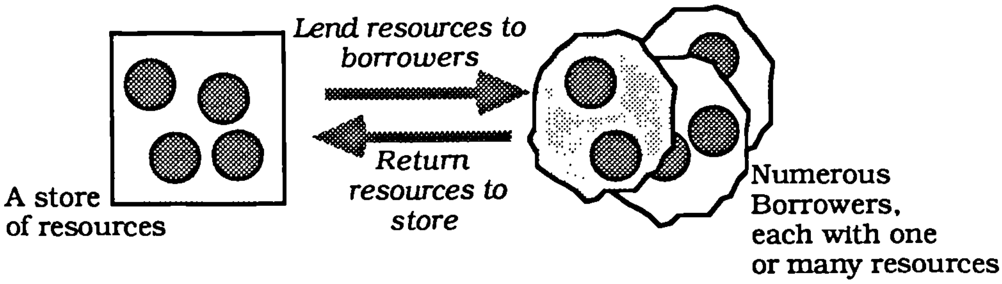

Example: This abstraction supports an analogy between domain for recording personnel starting and leaving an organisation, and for recording information about students beginning and leaving a university course. Both domains have a static space (organisation/course) which contains many objects (staff/students) at any time.



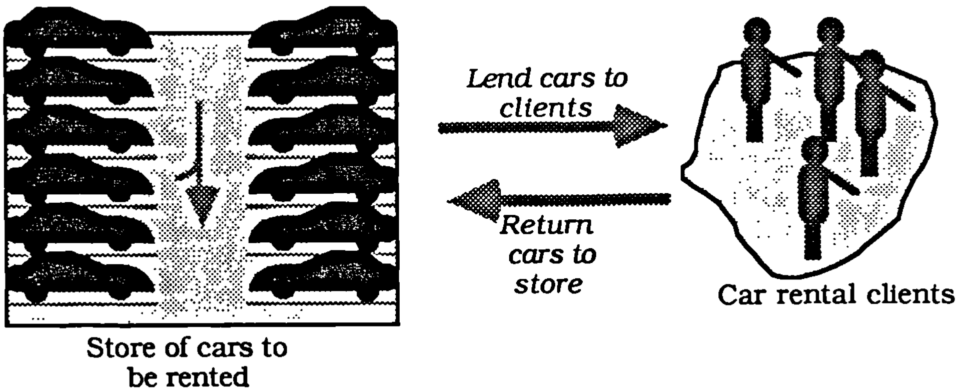



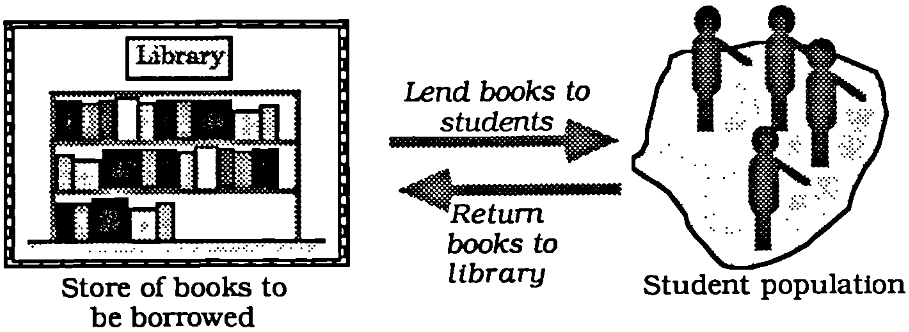
### 5: Non-Renewable Resource Management

Resources held in a store are lent to borrowers and then returned at a given date or time. The resource store is self-renewing, and not replenished from other sources within the scope of this abstraction.



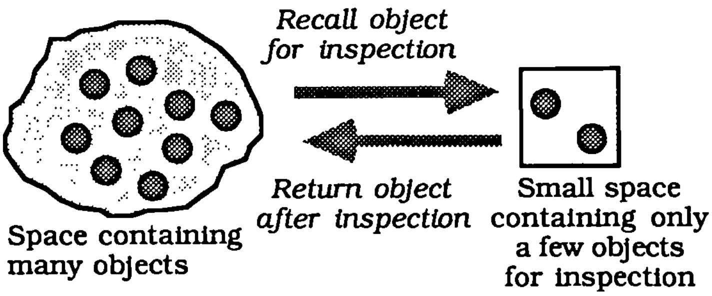

Example: This abstraction supports an analogy between car rental and university library domains. Both domains have a store of resources which are hired/lent to borrowers who normally return them to the store, so each object is normally lent and returned many times.



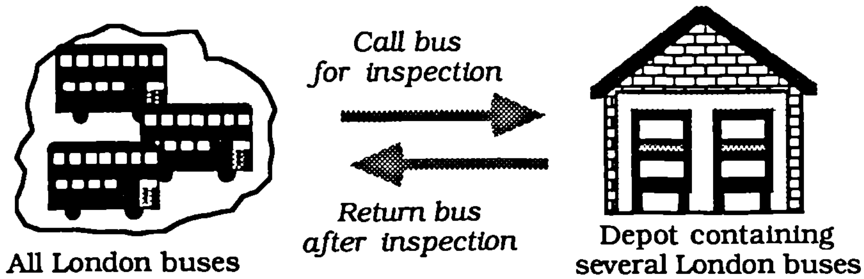



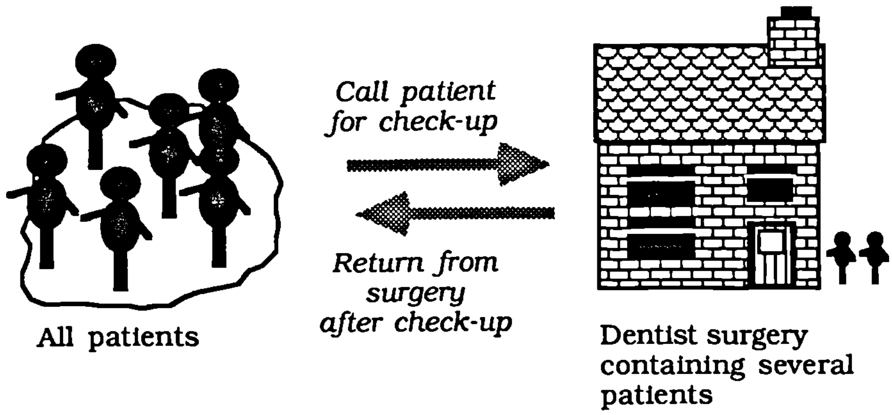
## 6: Periodic Object Recalling

Many objects exist in a space and most are periodically called to a space for inspection or other purposes. Inspection periods are most often temporal or linked to usage of the object. In exceptional circumstances inspections may be postponed and rearranged if the initial inspection cannot be attained. One or many objects can be inspected at a time.



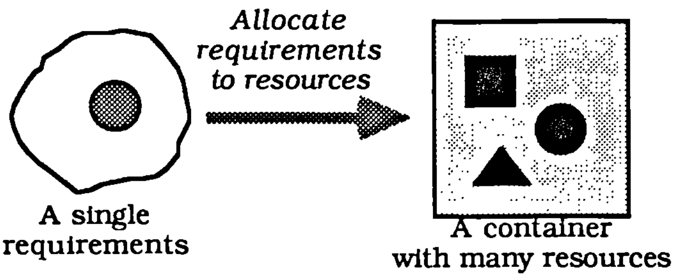

Example: This abstraction supports an analogy between domains for servicing the heavily-used London buses and for regular dental checkups at a surgery. Buses are returned to the maintenance depot after a given mileage (determined by the age of the bus) for a complete service and overhaul. This servicing is managed to ensure that buses are quickly back on the road and that the depot does not have an overwhelming number of buses to service at any given time. Similarly patients are called to a dental surgery every six months for a regular checkup which may be followed by special care if necessary. If necessary servicing of buses and patients' mouths can be rescheduled if initially inconvenient.



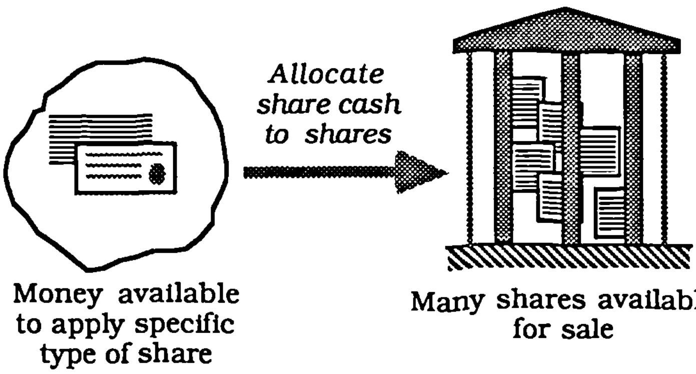



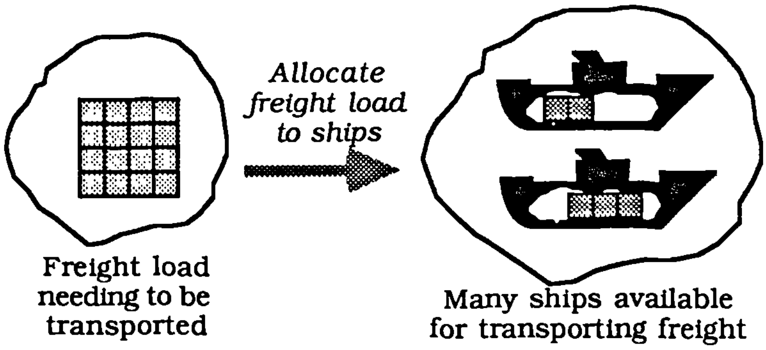
## 7: Single Object Allocation

Single object allocation is more complex and involves the allocation of a single requirement to resources if some constraints are met, i.e. the requirement and allocated resources share the same properties.



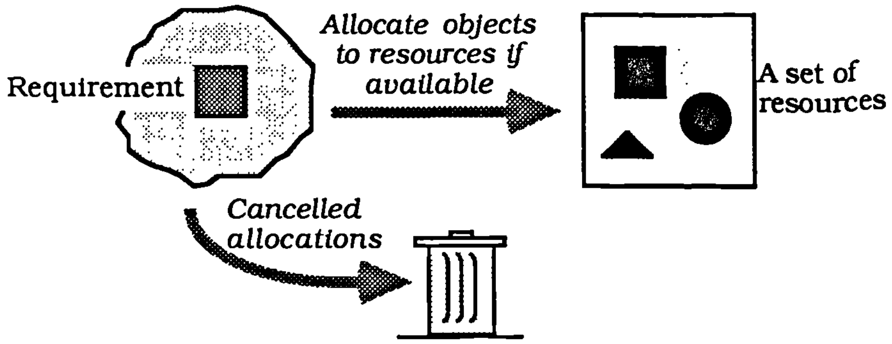

Example: This abstraction supports an analogy between domains for buying the most appropriate shares for a small shareholder at a share shop and for selecting the most appropriate ship with which to transport a cargo to a foreign port. In both cases allocation is constrained by the availability of appropriate shares or shipping holds.



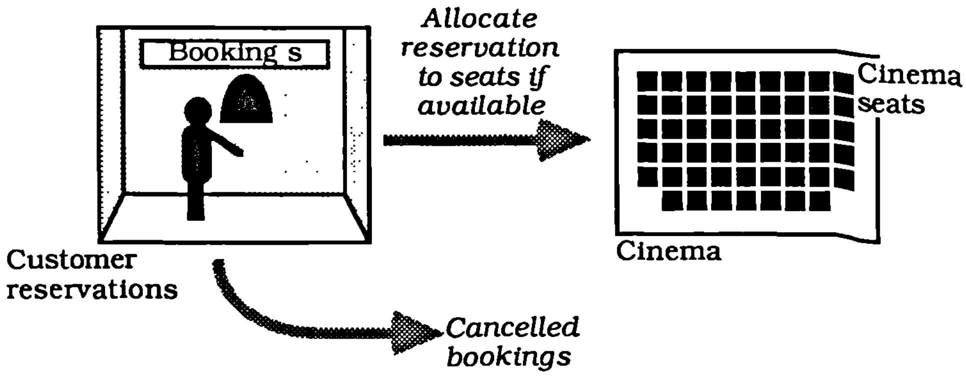



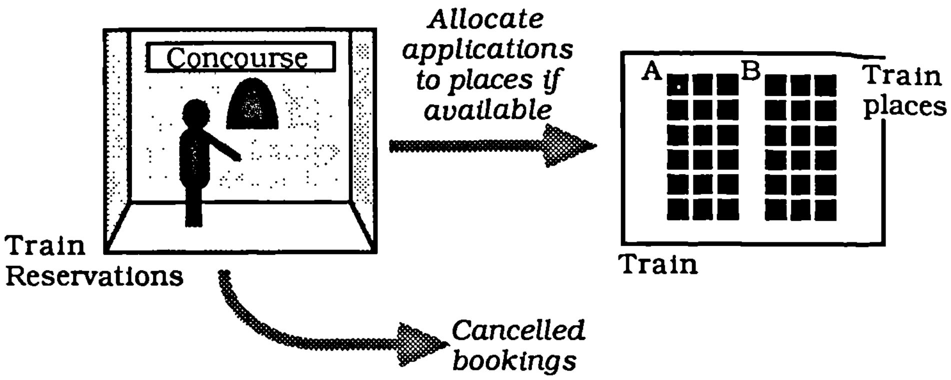
## 8: Single Object Allocation, no Waiting List

A single set of requirements are allocated to a larger set of resources if a paired requirement and resource share the same properties. If allocation of an object cannot be achieved due to no available reosurces, they are discarded.



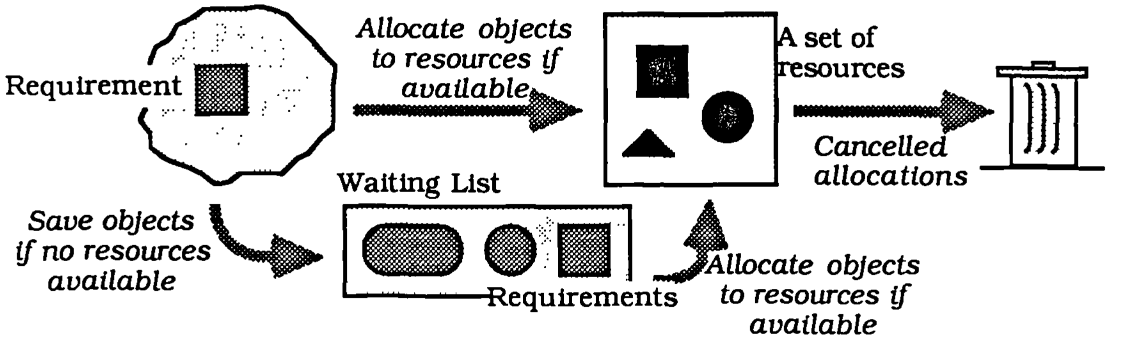

Example: This abstraction supports an analogy between local cinema seating and reserving places on an intercity train. Both domains have requirements (fulfilment of seats) which are met by resources (seats) if they meet constraints, for example seats are non-smoking in the cinema or on the train, expensive or cheap seats etc.. In both cases, waiting lists are impractical so unmet cinema or train requests are not kept but are discarded.



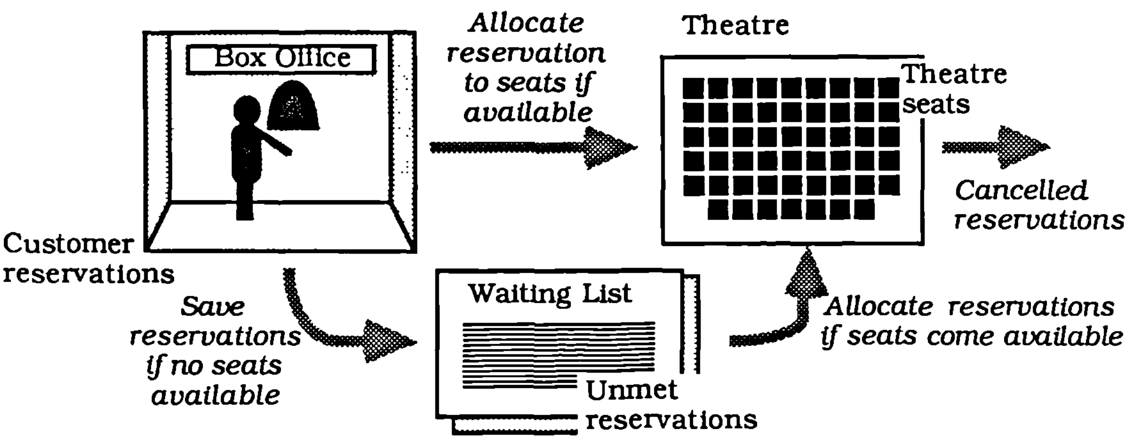



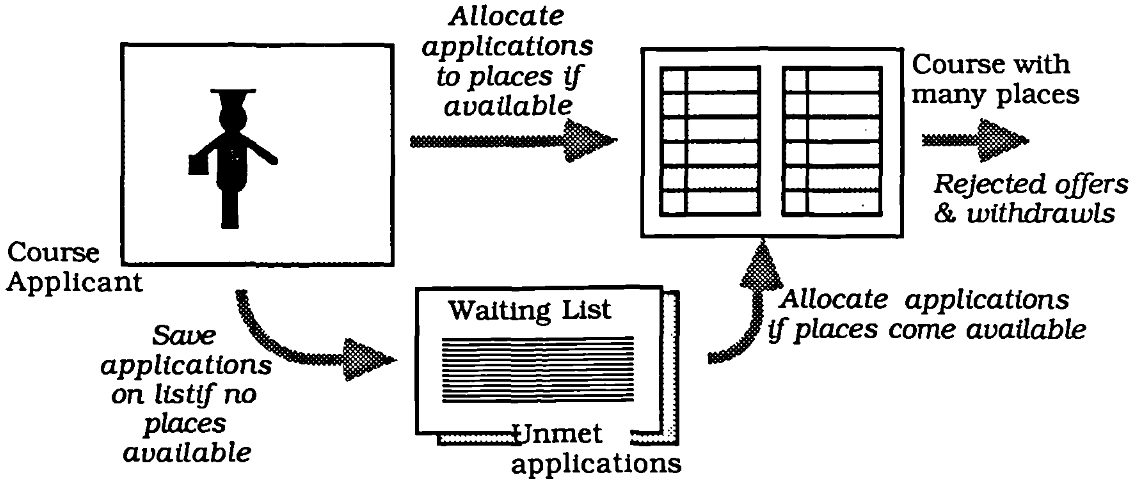
## 9: Single Object Allocation, with Waiting List

A single set of requirements are allocated to a larger set of resources if a paired requirement and resource share the same properties. If allocation of an object cannot be achieved due to no available reosurces, requirements can be reserved on a sequential waiting list. Subsequent cancellations of resource allocations can be fulfilled from the waiting list. Requirements on the waiting list are prioritised, and requirements on the list have priority over newly-arrived requirements.



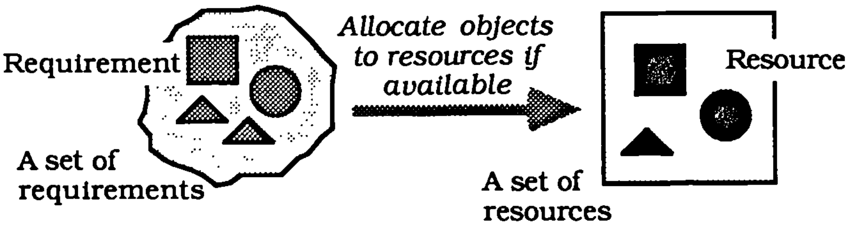

Example: This abstraction supports an analogy between theatre reservation and university course adminstration domains:



## 10: Multiple Object Allocation

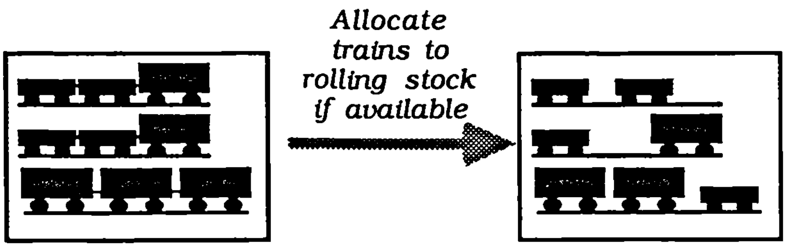
A set of many requirements are allocated to a larger set of resources if a paired requirement and resource share the same properties which act as constraints on the matching process. The aim of the computerised allocation process is to maximise allocation of requirements to resources, so resources are fully used and no requirements are left unfulfilled.




Example: This abstraction supports an analogy between domains involving staff allocation to new projects within an organisation and allocating freight stock to compose goods trains needed on British Railways.


Required new projects, each needed many staff

Project staff

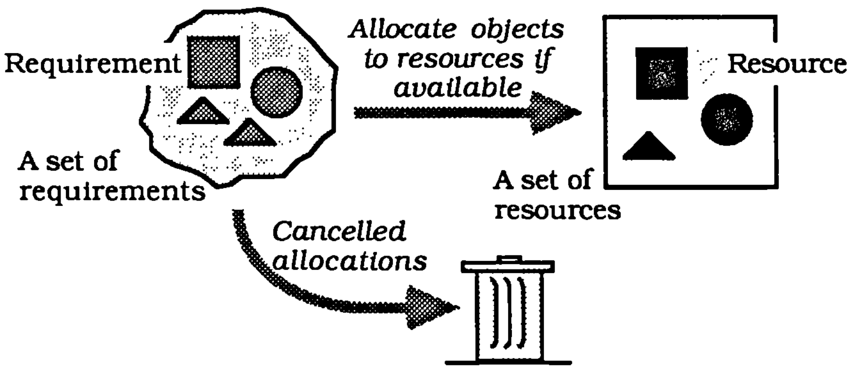



Required freight trains

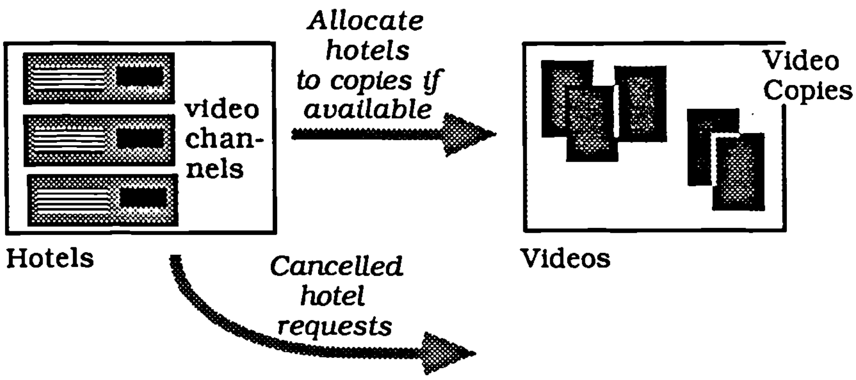
Rolling stock available to compose trains

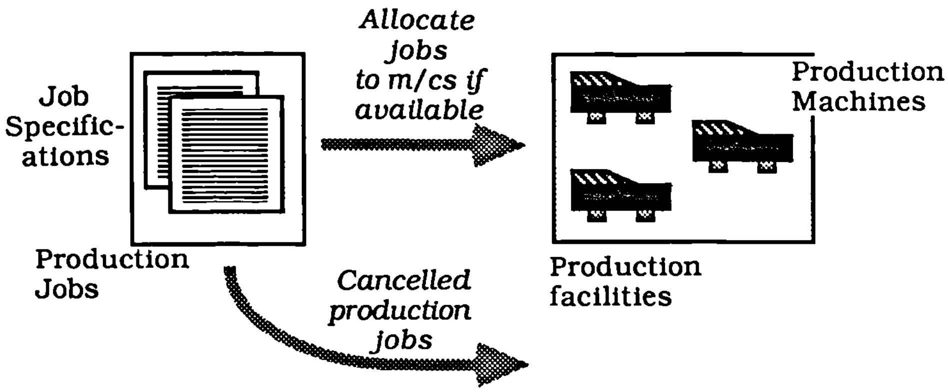
## 11: Multiple Object Allocation, No Waiting List

A multiple set of requirements are allocated to a larger set of resources if a paired requirement and resource share the same properties. If allocation of a requirement cannot be achieved due to no available reosurces, the requirement is rejected from the allocation.



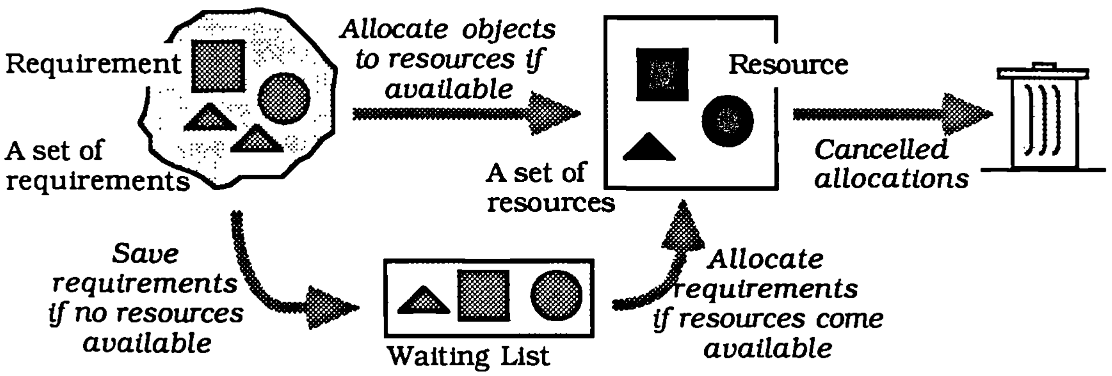

Example: This abstraction supports an analogy between video hiring and production planning domains. Both domains involve the allocation of groups of requirements to a set of resources which fulfil these requirements. If required production jobs or hotel needs cannot be met by machines or video copies respectively they are rejected as unfulfilled.



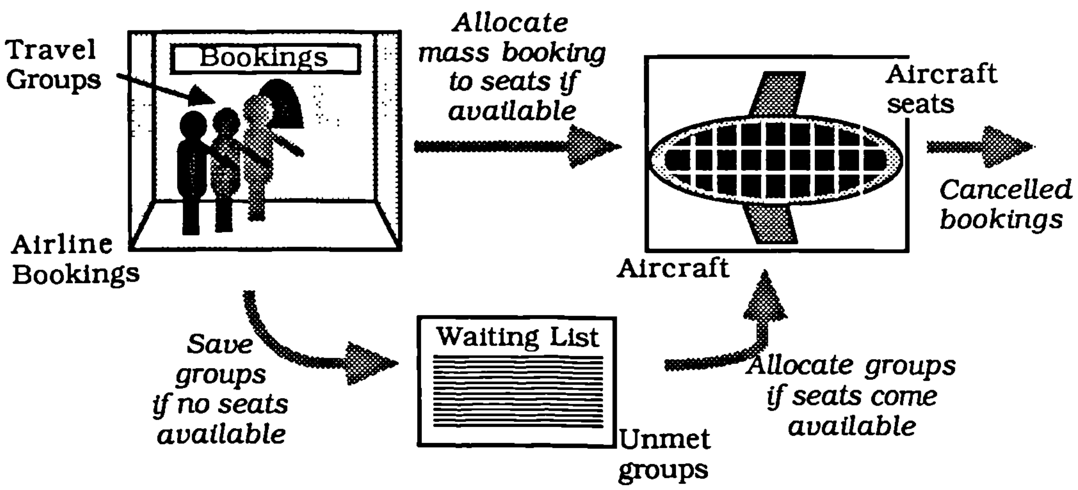



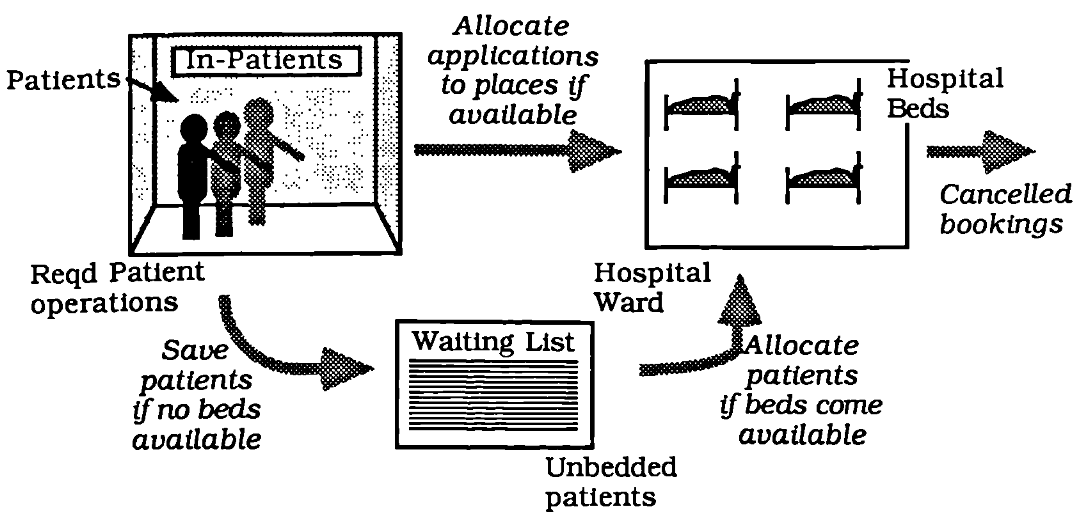
## 12: Multiple Object Allocation, with Waiting List

A multiple set of requirements are allocated to a larger set of resources if a paired requirement and resource share the same properties. If allocation of an object cannot be achieved due to no available reosurces, they are placed on a prioritised waiting list are allocated from the list to resources are come available when allocated requirements are cancelled.



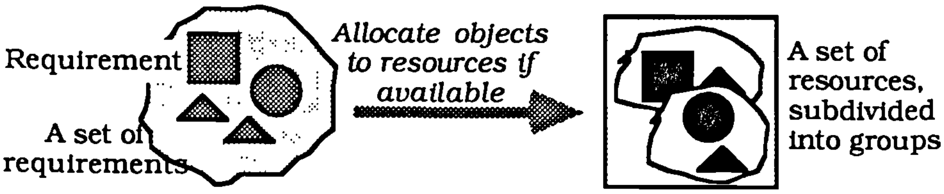

Example: This abstraction supports an analogy between domains for mass booking of airline tickets (e.g. charter trips) and planning for allocation of hospital beds for routine operations. Both domains have grouped requirements (fulfilment of batches of seats) which are met by resources (seats). In both cases, if patient or travel bookings cannot be met, they are placed on waiting list in case allocations of existing bookings occur.



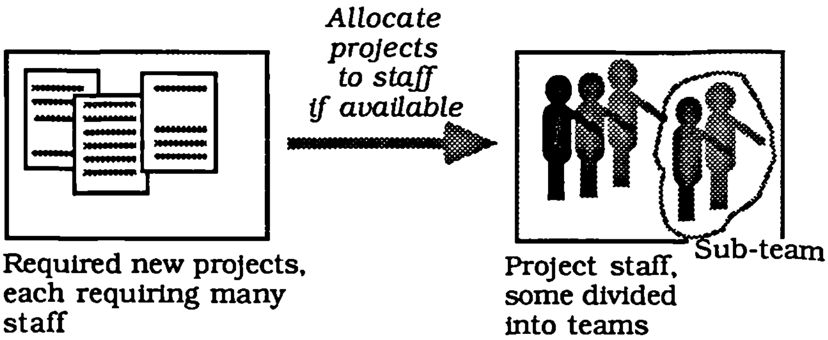



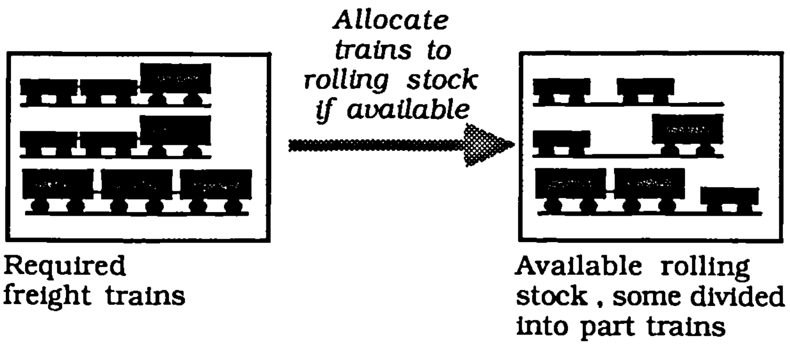
## 13 Object Allocation, Grouped Resources

A set of requirements are allocated to a larger set of resources if a paired requirement and resource share the same properties. Resources are grouped to provide larger resources for meeting more complex requirements. Allocation is optimised if resources can be exploited in groups otherwise groups must be broken down into smaller resources before being allocated. Subgrouped resources can occur for multiple or single object allocation domains with or without waiting lists for unmet allocations.



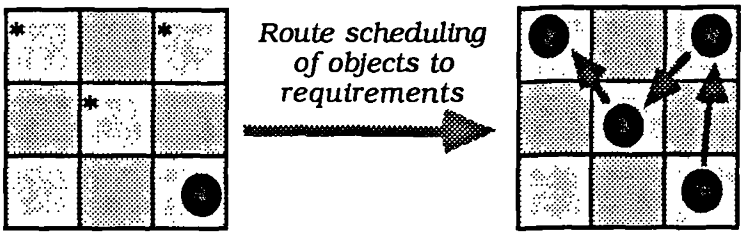

Example: This abstraction supports an analogy between domains involving staff allocation to new projects within an organisation and allocating freight stock to compose goods trains needed on British Railways. In both domains requirements (project members or rolling stock) may be grouped into subsets (e.g. project teams or preassembled trains) which can be allocated on mass, thus simplifying and improving the benefits from the allocation.



## 14: Route Scheduling

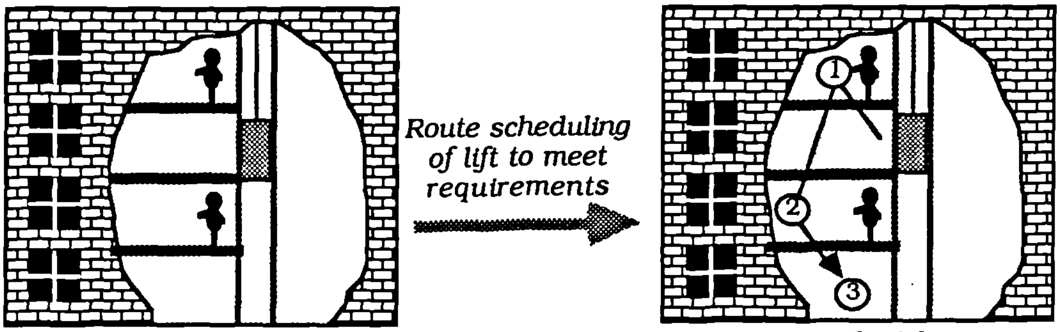
In route scheduling domains objects move between spaces to meet a set of predetermined requirements. Routes between these spaces can be optimised to minimise travel time or length while still ensuring all requirements are still met. This domain type is similar to object allocation domains with the addition of a temporal element, in that both domains have constraint satisfaction as a central issue.




Starting position & required positions to be met (*)

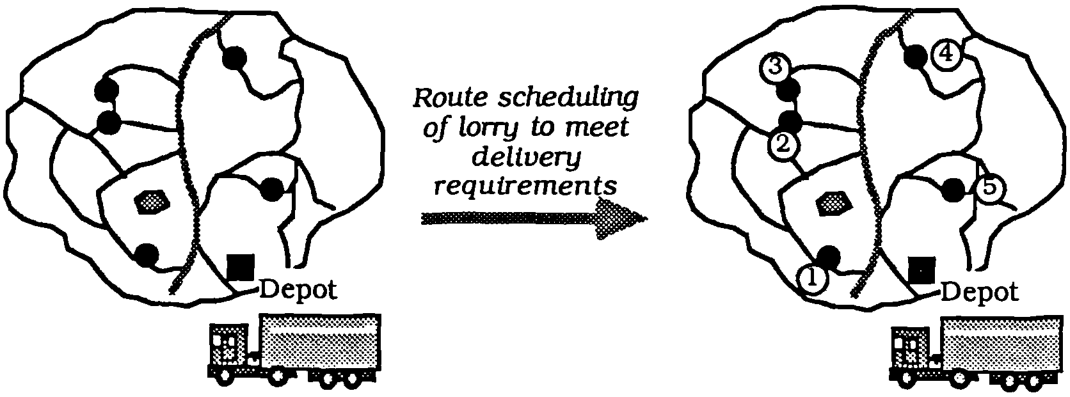
Final position demonstrating final route

Example: This abstraction supports an analogy between domains involving the management of a lift serving a small appartment block and route planning of lorries making deliveries to customers on a daily basis. Both domains have a set of constraints (e.g. destination floor, waiting time, order due-by-date ) which limit options for scheduling routes for the object (lift or lorry). The aim of both systems is to optimise route length and time for both the lift and each lorry.




Starting position & required positions of the lift

Route for lift to meet its requirements

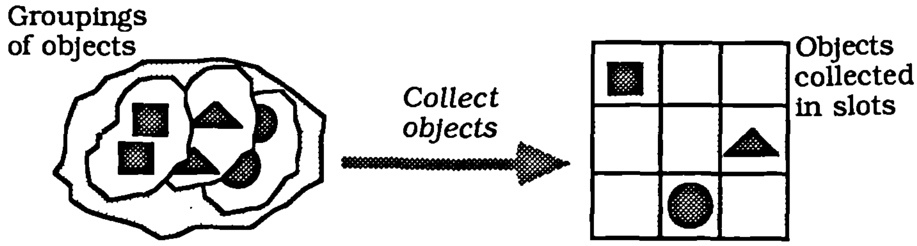



Starting position and required positions for the delivery lorry

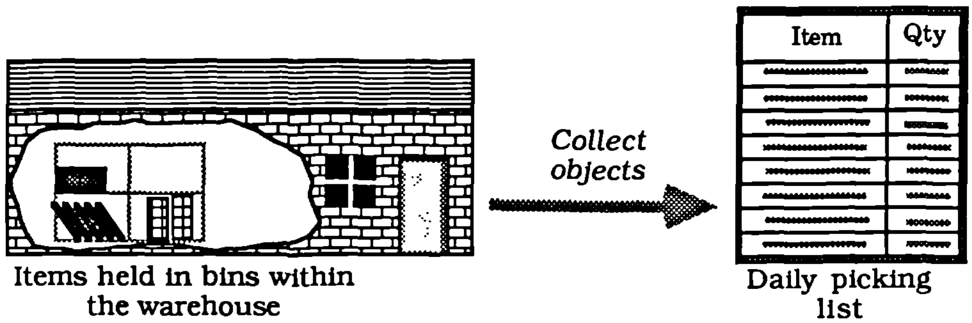
Route for lorry deliveries on the day

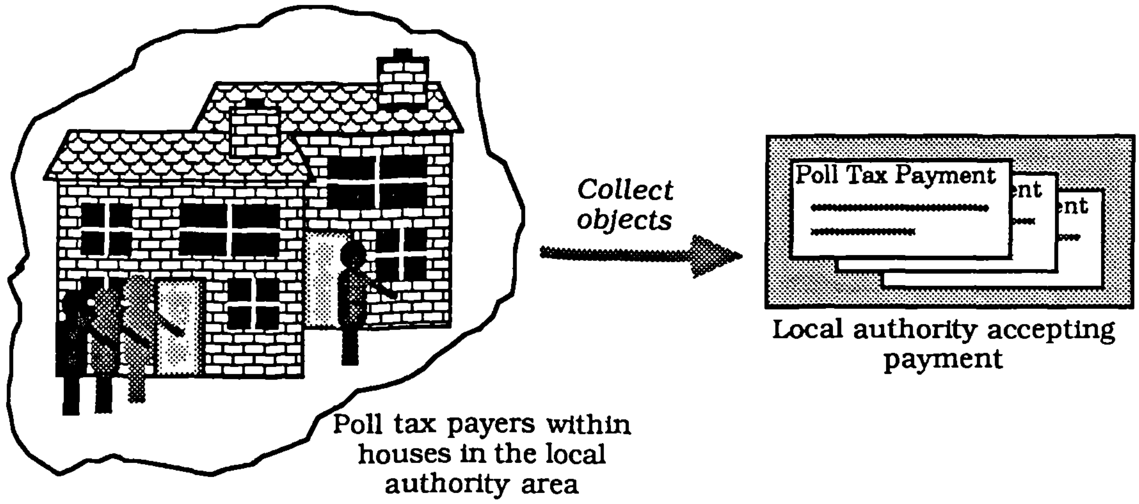
## 15: Manual Collection or Allocation

Objects are collected (or allocated) to sets by moving them from the source group to the slots. At any time there can be many objects in one or many sub-groups which must be collected to the current slot, so the concept of checking is included to ensure that collection is correct. This abstraction differs from that of the Allocation domains in that collection is beyond the scope of the information system while Allocation is carried out by the information system.



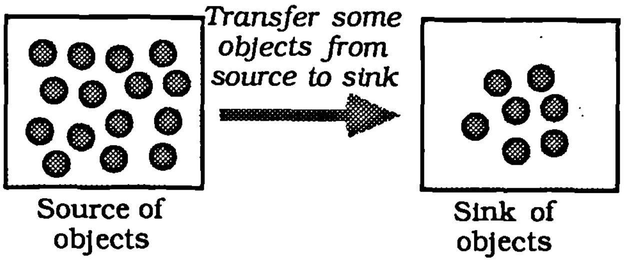

Example: This abstraction supports an analogy between order fulfillment picking and poll tax (or community charge) collection domains. Both domains have a large number of objects (poll tax payments or order items) which must be manually collected. These collections may be incorrect, so they are validated by the information system upon collection and before placing them in the slots (bins or payment).



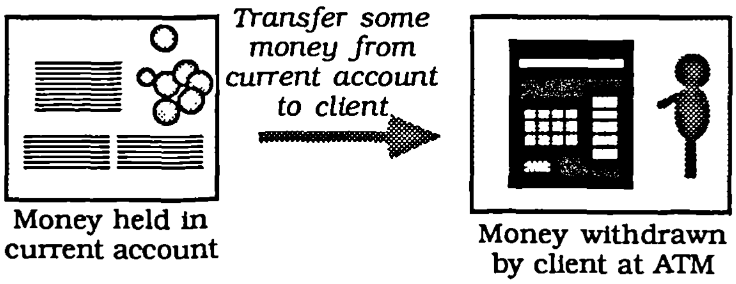



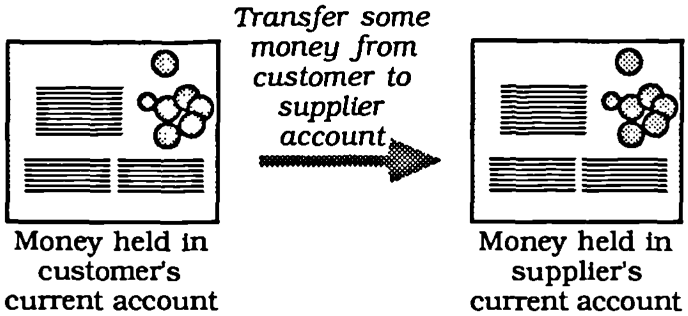
## 16: Commodity Transfer

A source space and sink space each contain varying amounts of an object. Specified quantities of this object move from the source to the sink space as shown in the figure below, reducing and increasing the levels of the object in the source and sink slots.



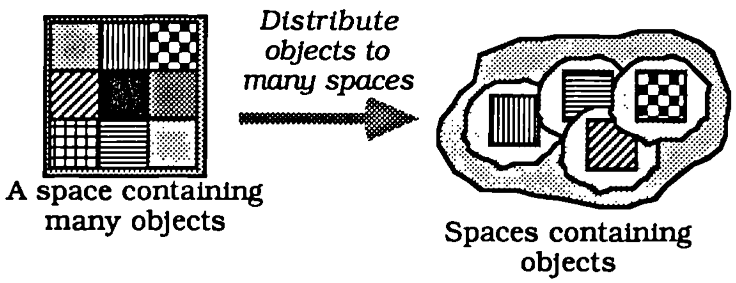

Example: This abstraction supports an analogy between financial transfer domains such as withdrawl of money from a current account using an ATM, and monthly transfer of order payment between a raw materials supplier and its customers. Transfer will not occur in both applications at any time if there is insufficient objects in the source to satisfy the transaction.



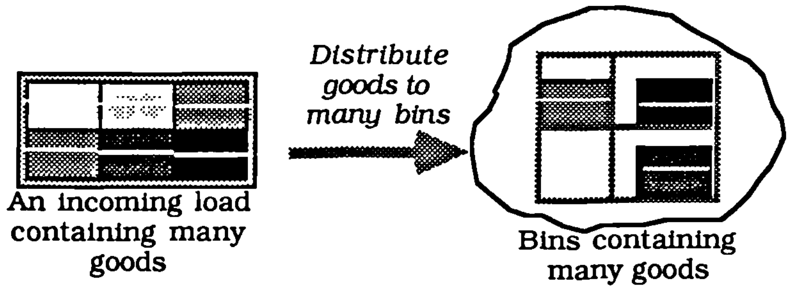



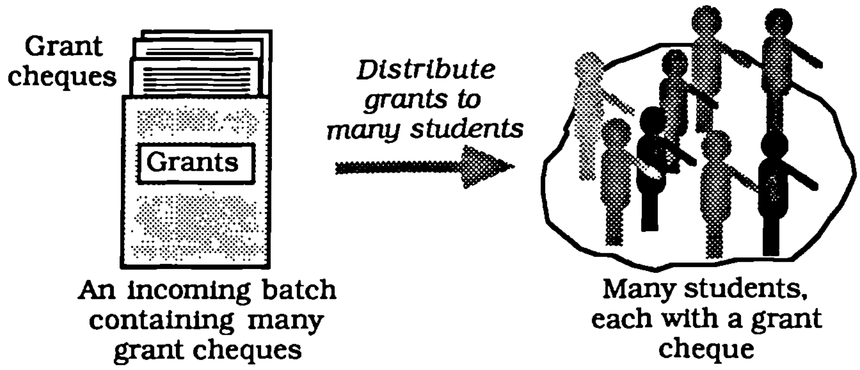
## 17: Object Distribution

Assembled objects are distributed to individual sources as specified in predetermined requirements. Objects enter the system on mass and are then distributed to their destinations. In exceptional circumstances objects can be sent to the incorrect destination, in which cases they may be returned to the destination.



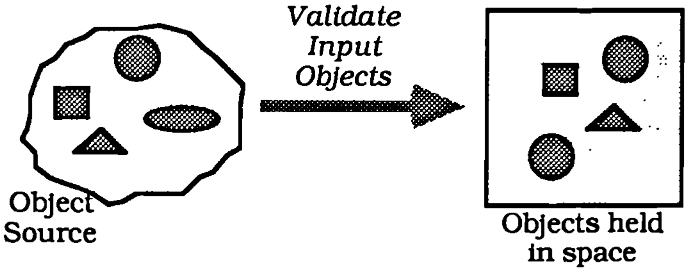

Example: This abstraction supports an analogy between domains for goods received distribution and distributing student grants within a university at the beginning of every term. In both domains the goods/grants arrive in batch and must be given to many sources, each of which may or may not reach its destination. As such, in exceptional circumstances, objects may be returned to the original source.



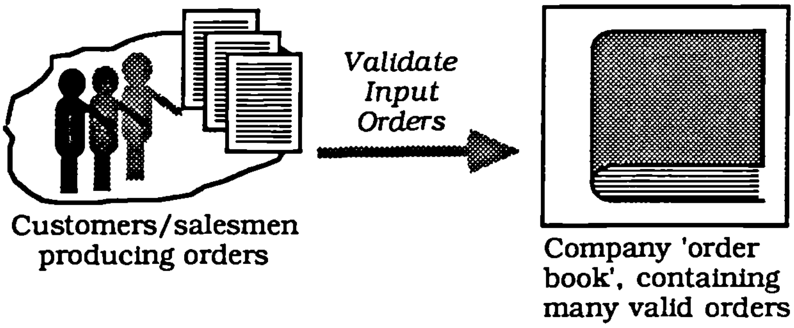



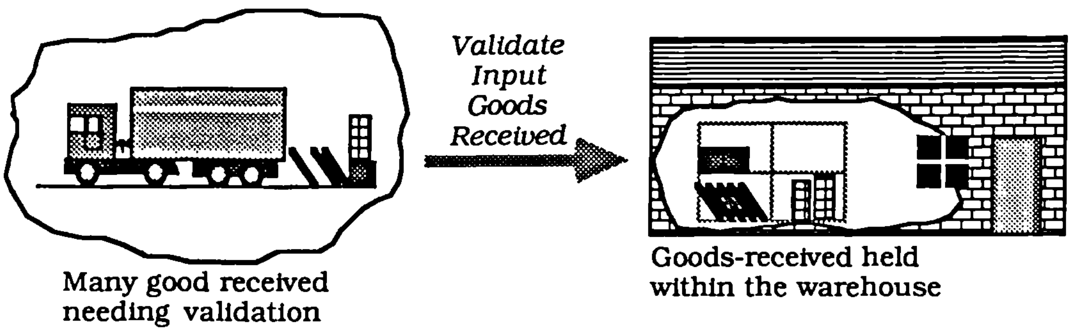
## 18: Input Object Validation

Objects are input to a space from the outside world if they meet specific validity constraints. In exceptional circumstances, if objects are incorrect or invalid , they are returned to their original source in the outside world. If valid they are moved to the space.



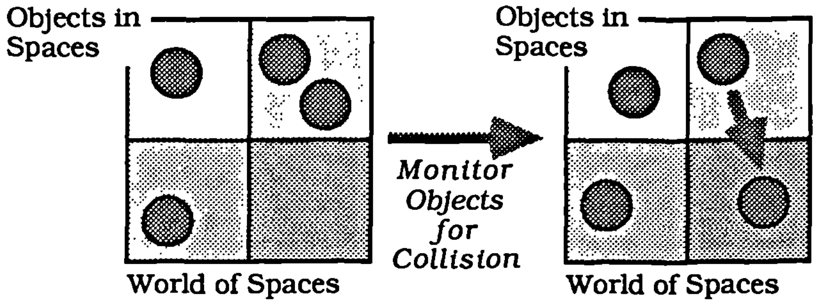

Example: This abstraction supports an analogy between sales order input validation and verficiation of goods-received from suppliers within a manufacturer's warehouse. If orders/goods fail to meet requirements they are returned to the suppliers or customers/ salesmen respectively. Orders are checked for validity, customer credit rating etc. while goods-received are also checked for breakages and defects, so in exceptional circumstances objects and goods-in do not enter the space.



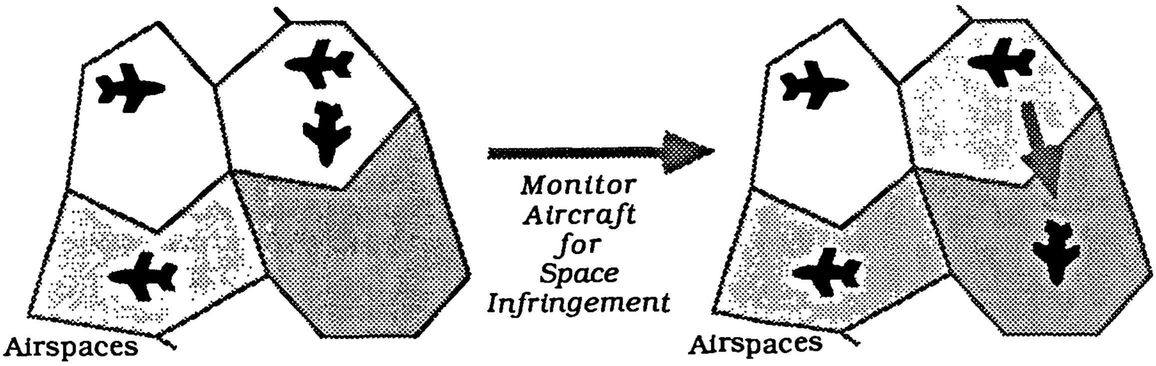



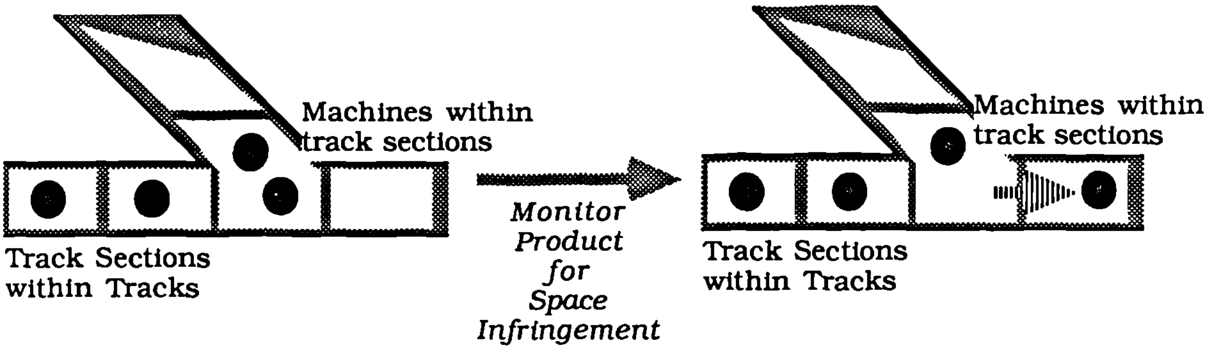
## 19: Object Monitoring

Many objects move in an area divided into many spaces, and move between spaces, although each object can only reside in one space at a time. They risk collision if two objects share the same space at the same time, so warning is necessary if any space simultaneously contains two or more objects. Correcting space violations is beyond the scope of the information system.



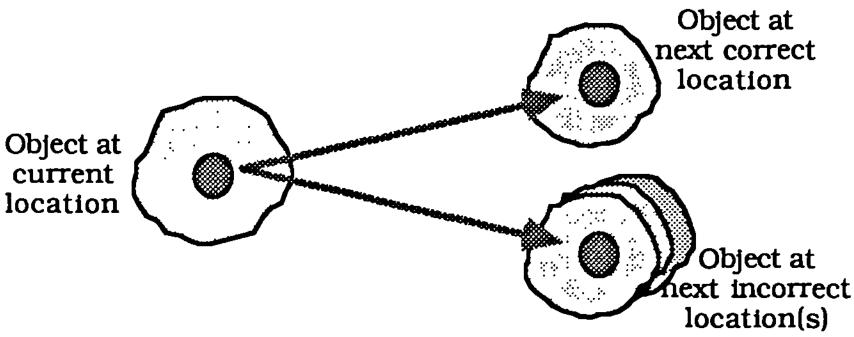

Example: This abstraction supports an analogy between air traffic control and flexible manufacturing domains. In both domains aircraft and products move and risk collision, so a system is required to warn of violations of an aircraft's air space and of a product's track sections respectively.



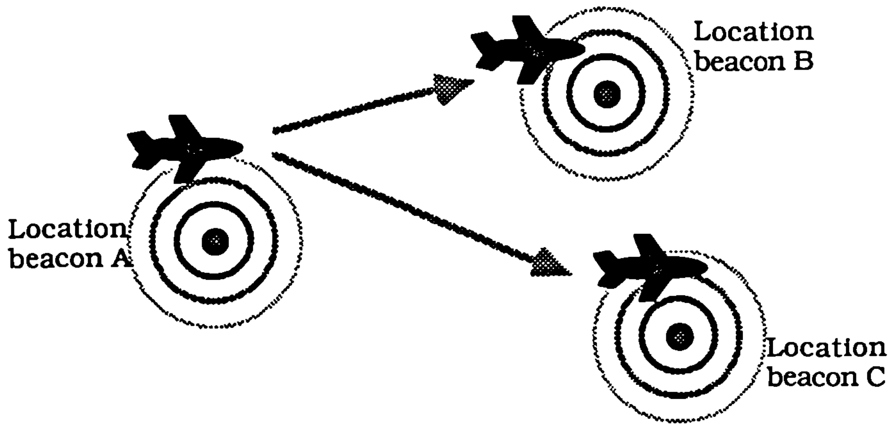



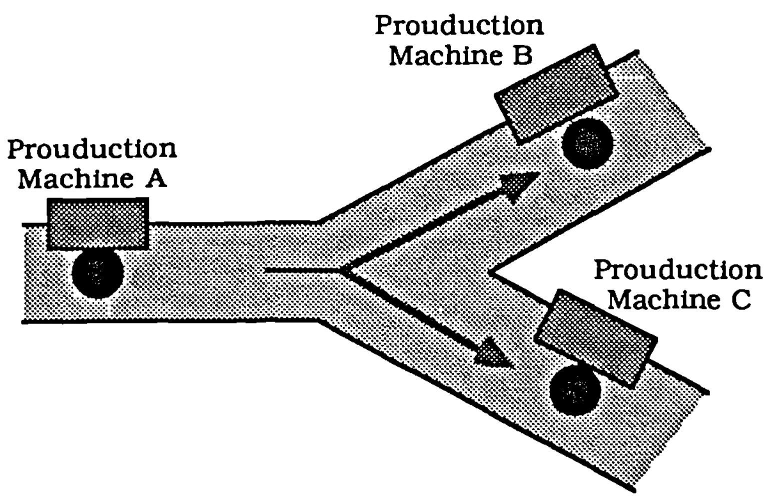
## 20: Plan Adherence

Objects move within a space and are guided by a predetermined plan which leads them through a series of temporal or positional steps. Objects can contradict plans and deviate from the predetermined steps, which must be corrected by prompting a warning or message leading to corrective action.



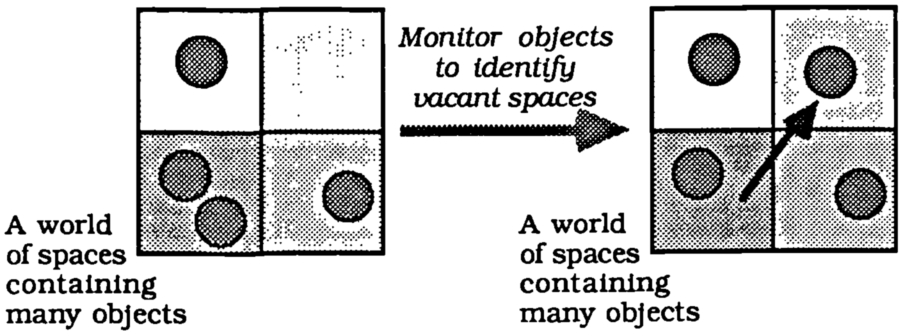

Example: This abstraction supports an analogy between domain for plan adherence during air traffic control and flexible manufacturing systems. In both domains aircraft and products move (freely) in spaces despite being constrained by steps laid down in the flight or production plans. Divergence from the plans is corrected elsewhere, but warning is initially necessary to warn of such divergences.



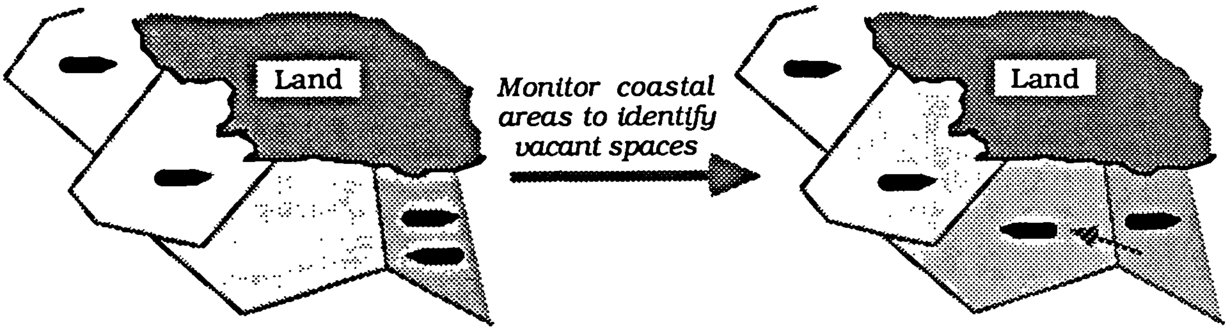



# 21: Object Positioning

Objects are positioned in a world divided into discrete spaces. The aim of the information system is to ensure that all spaces in the world are occupied by at least one object. Vacant spaces may occur because of object movement between or from spaces, so vacancies must be monitored for to provide warning for their correction, although correction itself is beyond the scope of the domain.




Example: This abstraction supports an analogy between coastguard patrol boat and modification of school timetabling domains. Both domains have objects (patrol boats or teacher monitors) which should be allocated to spaces represented as coatal areas or playtime sessions as appropriate. However, such initial allocations can change, so the domains must be monitored to identify and report vacant areas and sessions.




World of Patrol Zones containing many boats

World of Patrol Zones containing many boats

| Day | Morning | Lunch |
|---|---|---|
| Monday | *Smith/Hill* | *Jones* |
| Tuesday | *Peters* | *Maiden* |
| Wednesday | *Martin* | |
| Thursday | *Coates* | *Cooper* |
| Friday | *Smith* | *Sims/Bill* |

Timetable of playground supervision

*Monitor coastal areas to identify vacant spaces*

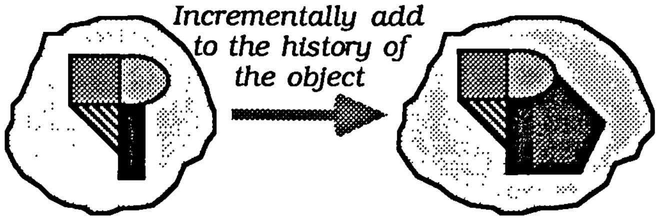
| Day | Morning | Lunch |
|---|---|---|
| Monday | *Smith* | *Jones* |
| Tuesday | *Peters* | *Maiden* |
| Wednesday | *Martin* | *Bill* |
| Thursday | *Coates* | *Cooper* |
| Friday | *Smith* | *Sims* |

*Sims*

Timetable of playground supervision

## 22: Object History Recording

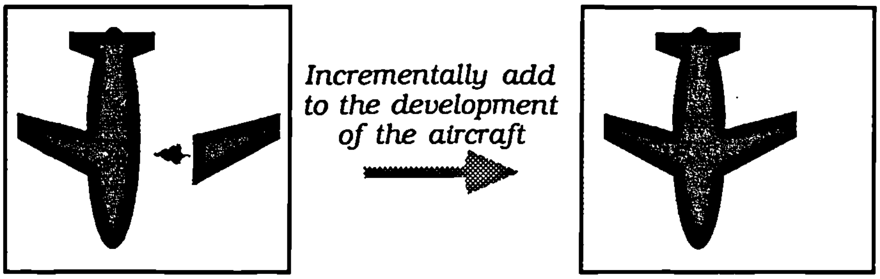
An object in a space undergoes a series of positive changes or additions to its state over time or geographic position. All changes to the object state are recorded by the information system.




Space with the initial object state

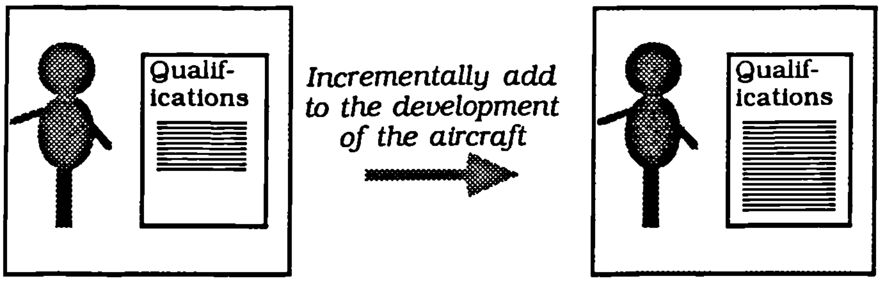
Space with the final object state

Example: This abstraction supports an analogy between domains for recording the history of development of an aircraft and the career development of each employee within an organisation. In both domains the object (aircraft/employee) gradually attain attributes representing their advancement within the organisation, either in terms of components (e.g. wing, cockpit) or achievements (e.g. grade 4 accoutancy exams). In the general case object accumulate attributes rather than loose them, although losses can occur in exceptional cases.




Aircraft in hanger without portside wing

Aircraft in hanger with portside wing




Employee in organisation with n qualifications & grading

Employee in organisation with n+m qualifications & grading

# Appendix B - Experimental Material for the First Empirical Study, Chapter 4

# Plumbline Problem Statement

## Foreword

I have prepared this extension to your original report, in order to meet a specific requirement which has since arisen, concerning Plumbline's requirements. I believe that it is imperative that you should consider the following, important task as your priority during this session.

Recent discussions with Mr O'Leary, the warehouse manager have highlighted the importance of improving our delivery and transportation system. The aim of this document is to provide you with specific details of the present delivery system, so that you may consider possible improvements, computer based or otherwise, concerning delivery operations.

From initial discussions with you and your colleagues I understand that computerisation of many of Plumbline's systems will take place. In your analysis should assume that the order entry and warehousing systems will be computerised. As such, work from the basis that input to the delivery system will include the forthcoming week's picking schedule, which idenifies all goods requiring delivery during that time.

## The Existing Delivery System

Deliveries are made to customers by both Leyland Diesel lorries and Ford Transit Vans. All of these vehicles are owned by Plumbline Discounts. Input into the existing delivery system is the pink Scheduling Note, sent directly from the orders clerk, and the goods from the warehouse, accompanied by the orange Delivery Note and the yellow Delivery Advice.

Presently vehicles are loaded according to a schedule worked out by Mr O'Leary. Mr O'Leary is very experienced in this task, having worked in the warehouse since the company moved to Clerkenwell, but his rule-of-thumb methods have sometimes been unable to cope with the recent growth in orders, leading to lorries starting out half empty, and ceratin orders being delivered late. Upon arrival at customer sites drivers have sometimes found that the required goods are at the back of the lorry, making it very difficult to retrieve them.

The increased turnover has in turn put a strain on the delivery fleet: at certain times employees have complained of the need for extra lorries and vans, although Mr O'Leary has dismissed this solution as cost-ineffective, based on the estimated level of expected orders. These problems have led to a certain personal friction between Mr O'Leary and Mr Tallboys, since it is Mr Tallboys whom customers complain to if their orders are late.

Due to the heavy workload of the Plumbline fleet vehicles sometimes breakdown. Mr O'Leary has often complained of not knowing of breakdowns until he has already prepared his daily schedules.

All pink Scheduling Notes are held on a spike by Mr O'Leary until delivery has been made, at which time they are filed in date of delivery order in ring binders in Mr O'Leary's office.

If customers refuse delivery of any orders or items then the orange Delivery Note is amended to show to amount actually delivered. Similarly requested items found to be missing upon delivery should be recorded, and an urgent order raised when the lorry returns.

Mr O'Leary has comlained to me recently that he is given inadequate warning of what orders are in the pipeline, and that if the pink Scheduling Note were received in Goods Out before the rest of the warehouse set then the delivery scheduling could be started earlier ( ie. before picking begins ). The computerisation of the warehousing system will provide such timely information. Indeed Mr O'Leary has expressed an interest in computers assisting him during the delivery scheduling task.

Mr O'Leary has also expressed an interest in scheduling deliveries, based on their required date. Both he and the sales manager, Mr Flynn, feel the existence of such information will benefit the Plumbline's systems in a number of ways. It will ensure that urgent orders are given priority during both picking and delivery. With such a system Mr O'Leary believes that it will be possible to develop a provisional weekly schedule, indicating all known deliveries required for the coming week, and a series of daily schedules, planning the daily deliveries of each vehicle.

Mr O'Leary has proposed the following format for a Daily Delivery Schedule:

| DAILY LOADING AND DELIVERY SCHEDULE | | | | | |
|---|---|---|---|---|---|
| Date: 24/10/88 | | Vehicle: C414 DEF | | Driver: N. Mansell | |
| Delivery Region: | | North London | | | |
| Customer | Address | Part No | Qty | Package | Description |
| Haringey Council | Central Office | X25413 | 004 | Boxes | Stand. Taps |
| | | A743 | 020 | Indiv. | Shank Toilets |
| | | D7693 | 001 | Indiv. | T3 Water Tank |
| | | S774 | 150 | Boxes | Stand. Washers |
| J. Bloggs, Plumbers | 2, Muswell Hill Broadway, N10 | X25413 | 001 | Box | Stand. Taps |
| | | E3481 | 006 | Crates | 12" Piping |
| | | W1089 | 002 | Indiv. | Stand. Baths |

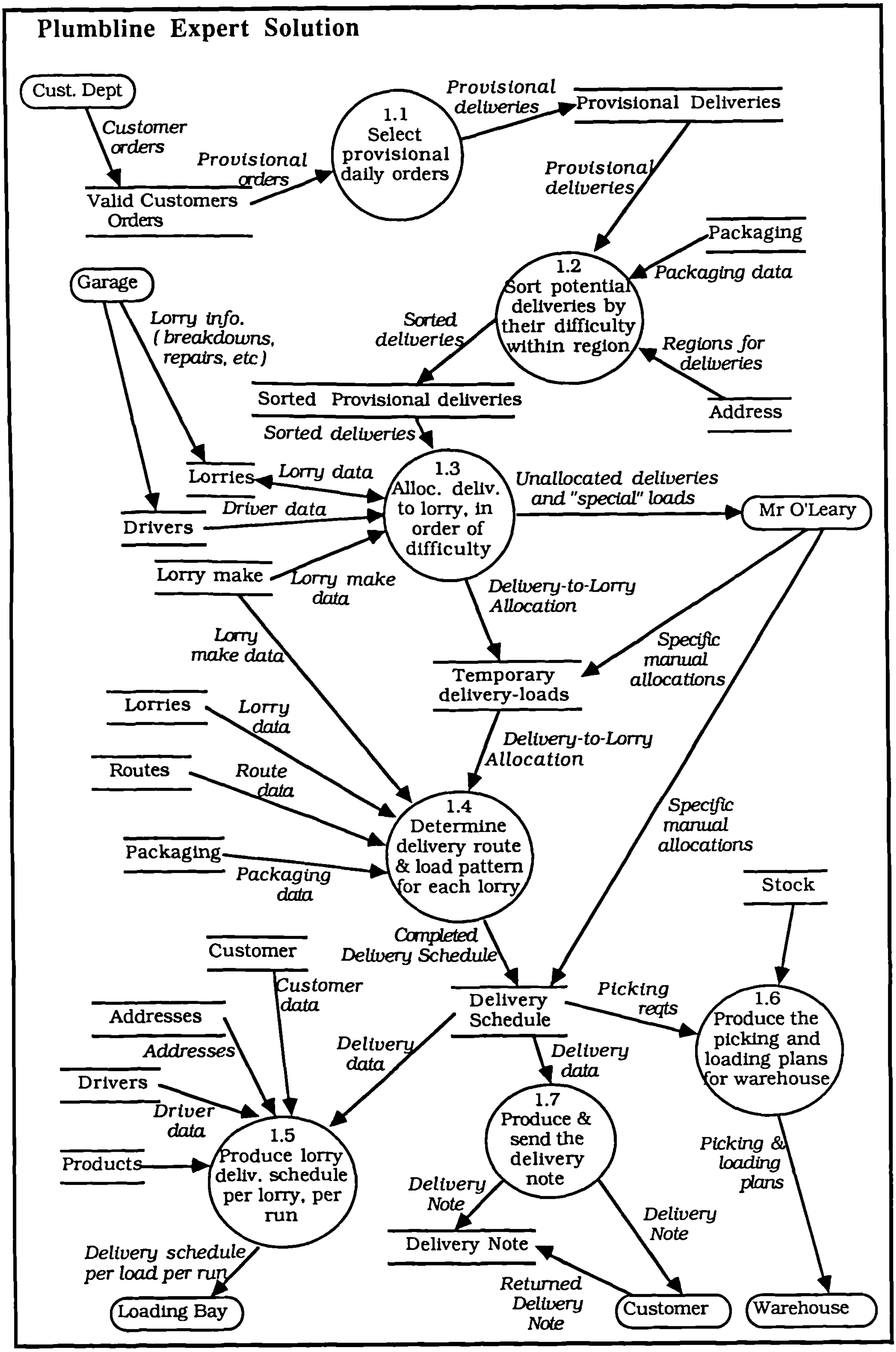
**Objectives of the Overall Study**

In order to ease your task and to save time I have repeated the major objectives of the initial study, outlined in your initial report:

- Reduce the turnaround time for Customers,
- Reduce the costs of order processing, warehousing and delivery,
- Reduce the paperwork and the time wasted in doing it,
- Distribute information around the company more speedily and accurately,
- Allow greater management control by providing more accurate and up-to-date information for regular and ad hoc queries,
- Provide management information by making available historic data in a form suitable for future decision making,
- Provide quicker and better scheduling methods.

I look forward to hearing your conclusions,

A. Noakes.


Plumbline Expert Solution
Cust. Dept
Customer orders
Valid Customers Orders
Provisional orders
1.1 Select provisional daily orders
Provisional deliveries
Provisional Deliveries
Provisional deliveries
Packaging
Packaging data
1.2 Sort potential deliveries by their difficulty within region
Regions for deliveries
Address
Sorted deliveries
Sorted Provisional deliveries
Garage
Lorry info. ( breakdowns, repairs, etc )
Sorted deliveries
Lorries
Lorry data
1.3 Alloc. deliv. to lorry, in order of difficulty
Drivers
Driver data
Lorry make
Lorry make data
Unallocated deliveries and "special" loads
Mr O'Leary
Delivery-to-Lorry Allocation
Specific manual allocations
Temporary delivery-loads
Lorry make data
Lorries
Lorry data
Routes
Route data
Packaging
Packaging data
Delivery-to-Lorry Allocation
1.4 Determine delivery route & load pattern for each lorry
Specific manual allocations
Stock
Completed Delivery Schedule
Customer
Customer data
Delivery Schedule
Picking reqts
1.6 Produce the picking and loading plans for warehouse
Addresses
Addresses
Delivery data
Delivery data
Drivers
Driver data
1.7 Produce & send the delivery note
Picking & loading plans
Products
1.5 Produce lorry deliv. schedule per lorry, per run
Delivery Note
Delivery Note
Delivery Note
Delivery schedule per load per run
Returned Delivery Note
Loading Bay
Customer
Warehouse

# Study 1 Checklist

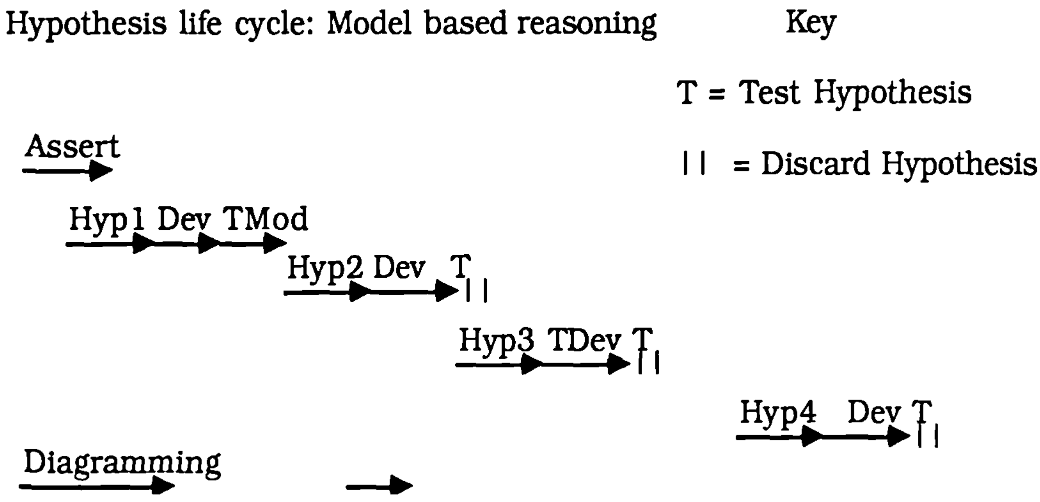
List of components checked for completeness in subjects' solutions. The list represents a combined solutions developed by 3 expert software engineers

| Components to be Included in the Specification | | In ? |
|---|---|---|
| Goals: | Improve existing delivery and transportation system | |
| | Partly computerise the loading scheduling function | |
| | Accept earlier computerised order information into the system | |
| | Define the necessary daily delivery route | |
| | Input breakdown knowledge into the system | |
| | Input refused order information into the system | |
| | Input information concerning missing delivery goods | |
| | Delivery scheduling should be based on order's required date | |
| | To develop a new daily delivery schedule | |
| | To develop a weekly delivery schedule | |
| | The scheduling sub-system must be flexible (m/m interface) | |
| data store: | A weekly schedule file | |
| | A daily schedule file | |
| | A vehicle file | |
| | A driver/mate file | |
| | An incoming orders file | |
| | A customer file | |
| | A delivery address file | |
| | A delivery-in-progress file | |
| | An archived delivery file | |
| Inputs: | Incoming sales orders | |
| | Information relating to the availability of vehicles | |
| | Information relating to driver/mate availability | |
| | Amendments to the returned delivery note | |
| | Input to create urgent orders | |
| | Override facility to permit human intervention in schedule | |
| Outputs: | Selection of most appropriate goods on each day | |
| | Efficient loading layout for the lorries | |
| | Route determination for most effective delivery route | |
| | Human check of delivery schedules | |
| | Update the delivery notes for missing/surplus goods | |
| | Identify possible problems for future week's deliveries | |
| Processes: | Select the appropriate goods for the appropriate day | |
| | Determine most efficient loading layout for the lorries | |
| | Route determination for most efficient delivery route | |
| | System may validate human-created schedules | |
| | Update the delivery notes for missing/surplus goods | |
| | Identify possible problems for the future week's deliveries | |

## Study 1 Example Protocol

| Behaviour | Protocol |
|---|---|
| Assert | Yeah, the first thing to notice is, that he has got two inputs from the orders dept, which is the actual pink scheduling note it says here, but it doesn't matter, he's just got some note from orders, ... |
| Create Hyp1 | so he needs some information from orders, and also he needs.... |
| Dev Hyp1 | obviously the goods from the warehouse as well, with some information again of what has been paid, and what's in the goods, so he's getting some information from the warehouse, and then, ... |
| Test Hyp1 | once those inputs are there, he, er, he's working out, then, schedule for delivery of goods, things that are ready, .. |
| Modify Hyp | so after he receives that information he does the schedule, ... |
| Create Hyp2 | um I assume that he does the schedule daily, so he, ... |
| Dev Hyp2 | he schedules everything for everything that he receives in the morning, or by sometime in the morning, say 10 O'clock, anything else that comes after, or at any, any orders, or anything from the warehouse that comes after 10, ... |
| Test Hyp2 | he would not schedule until next, er, next morning.... |
| Discard Hyp2 | no I think that's impractical. ... |
| Create Hyp3 | Er, but we must have some kind of clue as to when we have enough information to make a schedule... |
| Test Hyp3 | Er, because we don't need to say too much on the schedule until about 10 O'clock, it would seem that, ... |
| Dev Hyp3 | in order to make it to north London you need to send them, .. |
| Test Hyp3 | but that may include an empty van going, or half empty van going, so, |
| Discard Hyp3 | you need to, to make some decision about that, er, .. |
| Plan | I think that's something we will have to discuss with Mr O'Leary and the rest of the company,.. |
| Goal/Requ | of what kind of service do thay want to provide to their customers, or how critical it is that something goes morning or afternoon, or maybe the next morning instead. |
| Create Hyp4 | So we need to make a decision about when schedule occurs,.. |
| Dev Hyp4 | does it depend on time, or does it depend on the number of orders and goods we have to deliver ? Er, .. |
| Test Hyp4 | then, once the schedule is done, the lorries make the deliveries, er,... |

Figure 1a: Example of a protocol transcript showing model based reasoning - subject K.The duration of this segment is just under 5 minutes




Notes

The thematic link betweeen hypotheses 1-3 is planning delivery of goods, the need to schedule delivery and the information necessary to create a schedule. Hypothesis 4 returns to this theme although planning and goal recognition happen in the interim.

Figure 1b: Model of the sequential behaviour exhibited in Figure 1a by Subject K.

# Appendix C: Experimental Material for Second Empirial Study, Chapter 4:

- Target VI library problem and solution,
- Source Template Solution for Object Allocation Problem,
- Concrete Source Specification for Roker Manufacturing Systems

## Target: Video International Problem Statement

As a member of the analysis team charged with the development of Video International's systems, your role is to analyse and develop the Video-to-Hotel allocation function. **As a priority** you should :

**1** - Develop a JSD-type Entity Process Structure for the Video-to-Hotel allocation function,

**2** - Describe in any form you choose the process to most appropriately allocate the necessary video copies to each hotel. This process should ensure the maximum use of existing video copies, as well as meeting all hotel requirements,

**3** - List the constraints that should be applied to the Video-to-Hotel allocation function.

Secondly you should :

**4** - Identify other events which are important to the allocation of videos to hotels, possibly through the construction of other key, Entity Process Structures.

Initially VI intends to implement a simple computerised system. In this system copies of videos are allocated to hotels for a monthly period. Computerised video allocation should consider the following constraints when deciding whether a video can be allocated to a hotel:

- the video and television systems ( eg. VHS ) of each hotel must be consistent,
- ensure that the number of video copies available is sufficient,
- VI's distribution rights in the hotel's territory are valid,
- a particular video title should not be shown twice at a particular hotel.

Outside the scope for this computerised Video-to-Hotel allocation function are:

- 'pre-specified' barred videos for a particular hotel,
- linguistic and ethnic constraints,
- the film's distributor, and the film's length.

You should mention or introduce other, as yet unanalysed constraints to refine the function. The decision as to which constraints have priority will be made by the Distribution Manager: this order is not changed once the system is implemented. If a hotel's requirement cannot be met, based on existing video stock, new video copies should be requested from the distributors.

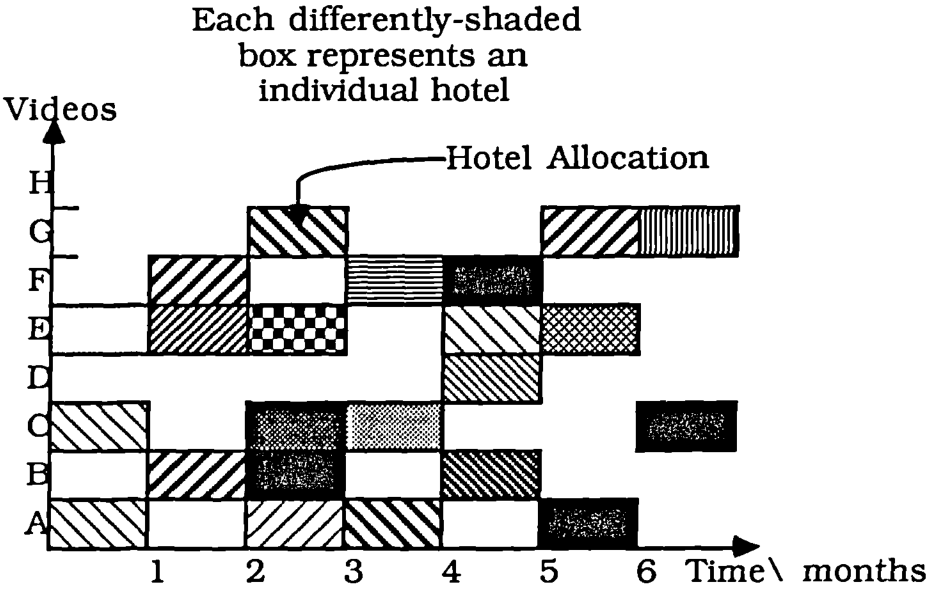
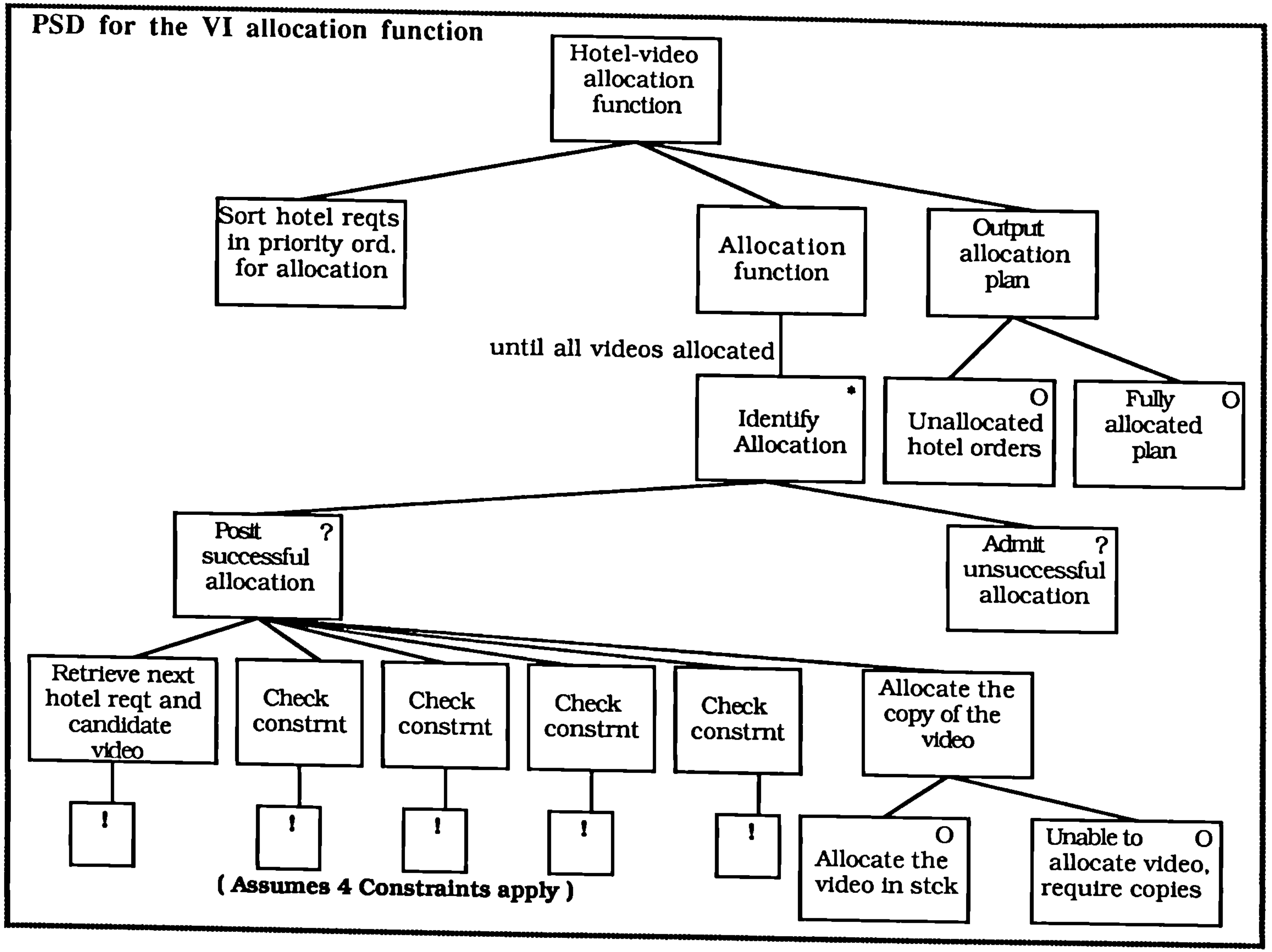



**Fig 1 - Video-Hotel Allocation Example**



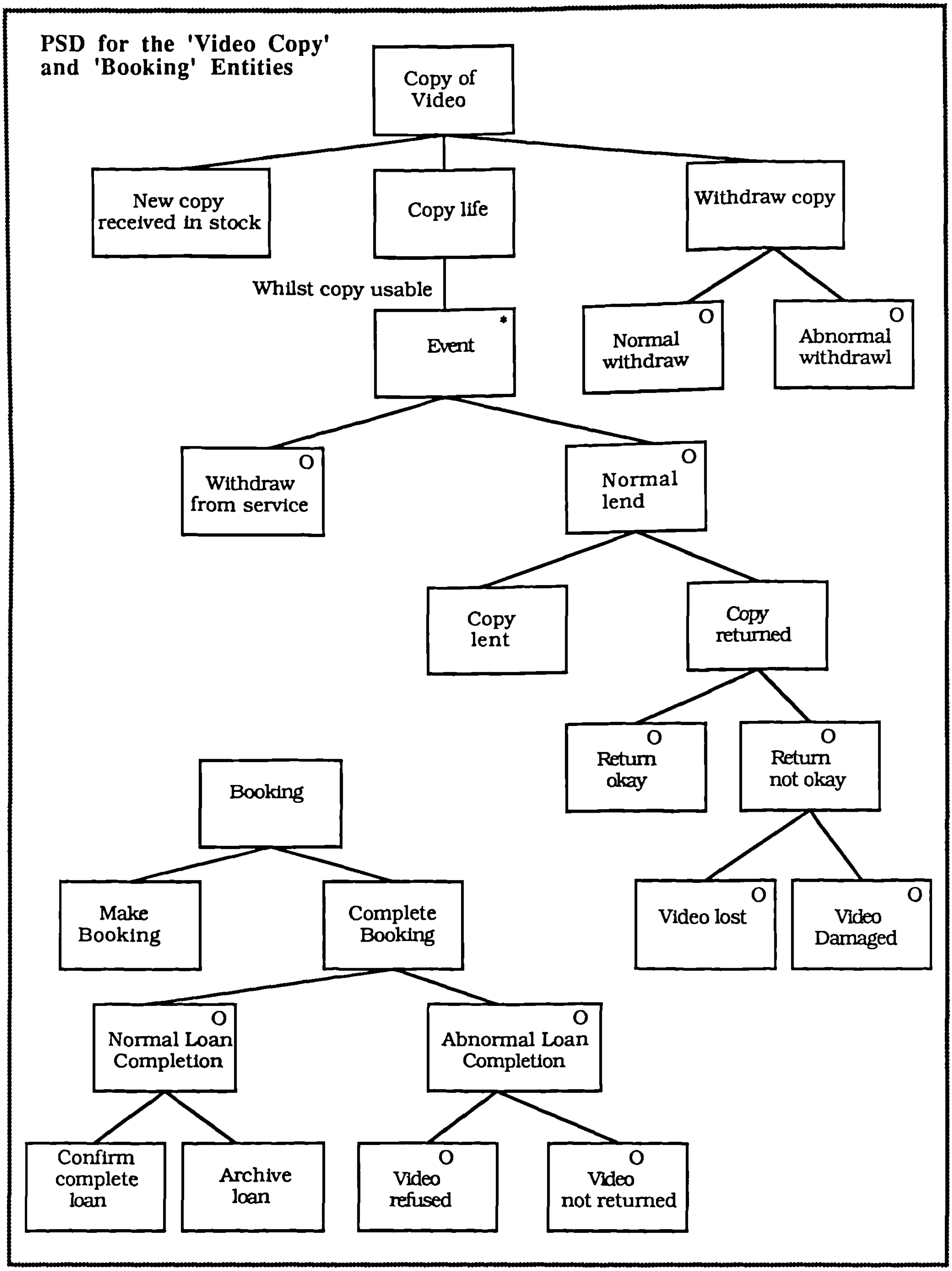
PSD for the VI allocation function
Hotel-video allocation function
Sort hotel reqts in priority ord. for allocation
Allocation function
Output allocation plan
until all videos allocated
Identify Allocation *
Unallocated hotel orders O
Fully allocated plan O
Posit successful allocation ?
Admit unsuccessful allocation ?
Retrieve next hotel reqt and candidate video
Check constrnt
Check constrnt
Check constrnt
Check constrnt
Allocate the copy of the video
!
!
!
!
!
Allocate the video in stck O
Unable to allocate video, require copies O
( Assumes 4 Constraints apply )

# Study 2 Checklist

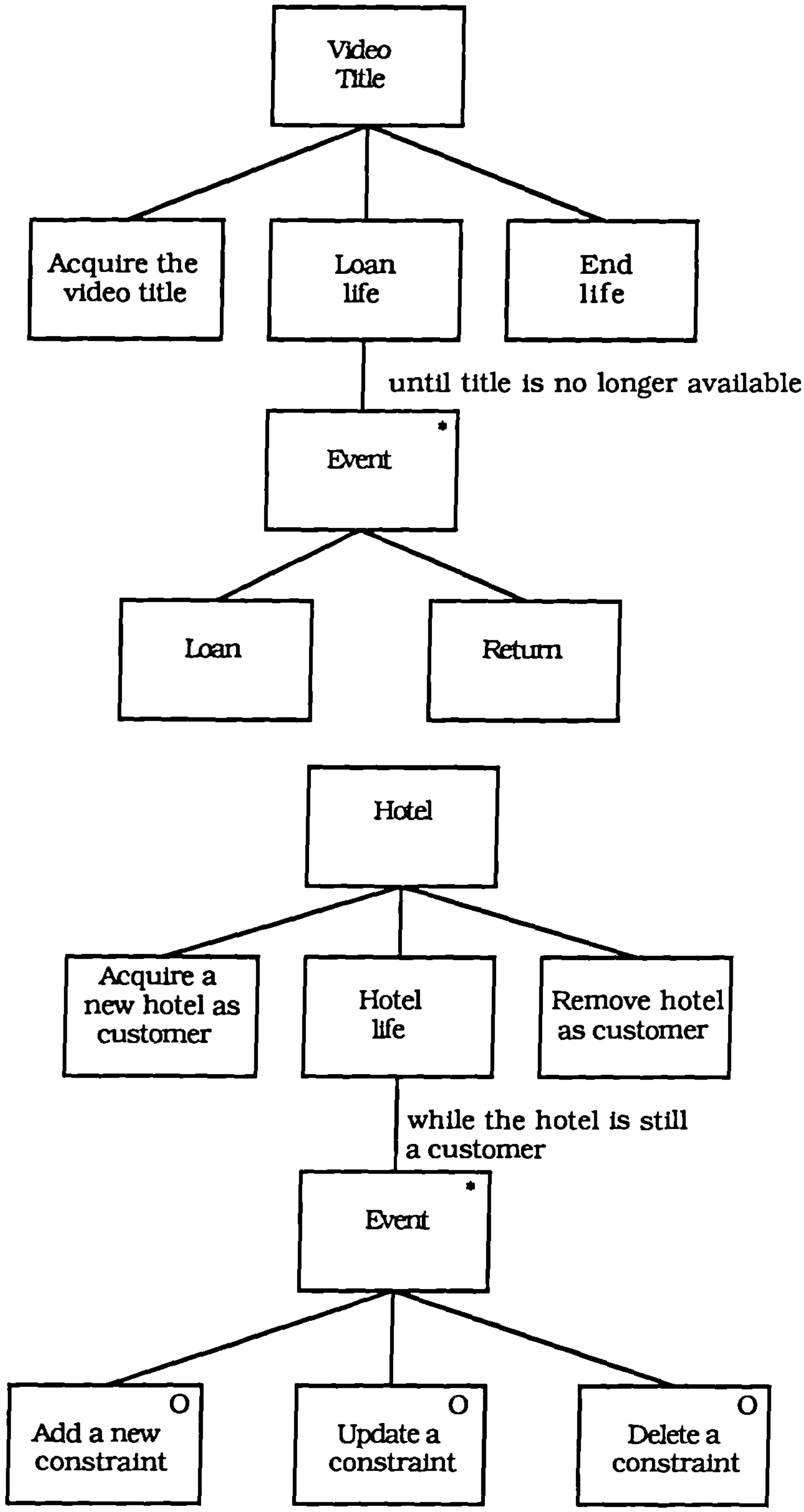
List of components checked for completeness and errors in subjects' solutions. The component list represents a combined solution developed by 2 expert software engineers

| | Completeness Checks | In ? |
|---|---|---|
| Components: | Sort hotels by their priority | |
| | Retrieve the next hotel/video to match | |
| | Allocate a video copy to a hotel | |
| | Allocate a video in stock to a hotel | |
| | Request new videos to be allocated to hotels | |
| | Output the allocation plan | |
| | Output the completed allocation plan | |
| | Output the partially allocation plan | |
| | Constraint: video systems must be consistent | |
| | Constraint: sufficient number of videos available | |
| | Constraint: distribution rights are okay | |
| | Constraint: same video must not be shown twice | |
| | Constraint: order the films shown at the hotel | |
| | Constraint: valid match under managment policy decisions | |
| | Constraint: film censorship ratings | |
| Structures: | An allocation sequence | |
| | An iterative allocation matching | |
| | A selection between a matched and unmatched allocation | |

| Errors Checks | | | |
|---|---|---|---|
| *Syntax* | | *Semantic* | |
| Confuse data stores and sources on Ctxt DFD | | Failure to identify major processes | |
| Missing data flow labels\arrow (>4) | | Determine wrong data stores for processes | |
| Misuse of other data flow DFD symbols | | Failure to recalculate air space | |
| Breaking DFD rules | | Combine P4 & P5 unnecessarily | |
| Failure to draw both DFDs | | Failure to recognise the problem scope | |
| | | Too many data stores for processes | |
| | | Source rather than target objects in spec | |


PSD for the 'Video Copy' and 'Booking' Entities
Copy of Video
New copy received in stock
Copy life
Withdraw copy
Whilst copy usable
Event *
Normal withdraw O
Abnormal withdrawl O
Withdraw from service O
Normal lend O
Copy lent
Copy returned
Return okay O
Return not okay O
Video lost O
Video Damaged O
Booking
Make Booking
Complete Booking
Normal Loan Completion O
Abnormal Loan Completion O
Confirm complete loan
Archive loan
Video refused O
Video not returned O

**PSDs for the entities Video Title' and 'Hotel'**



## Concrete Source Roker Manufacturing Production Planning System

The production planning function allocating required production jobs to available machines is described in more detail in the following paragraphs.

This allocation function is central to the production planning process. The relevant constraints, in no particular order, are :

- suitability of the machine to fulfil a job's manufacturing requirement,
- processing capacity of a machine,
- the number of machines available for use,
- safety and legal constraints imposed upon the use of a machine,
- machine operator availability for a machine,
- the Rating of the machine's speed to handle certain manufacturing job quantities,
- production management policy decisions on machine loading and working,
- the sequence in which manufacturing jobs on a product must be carried out.

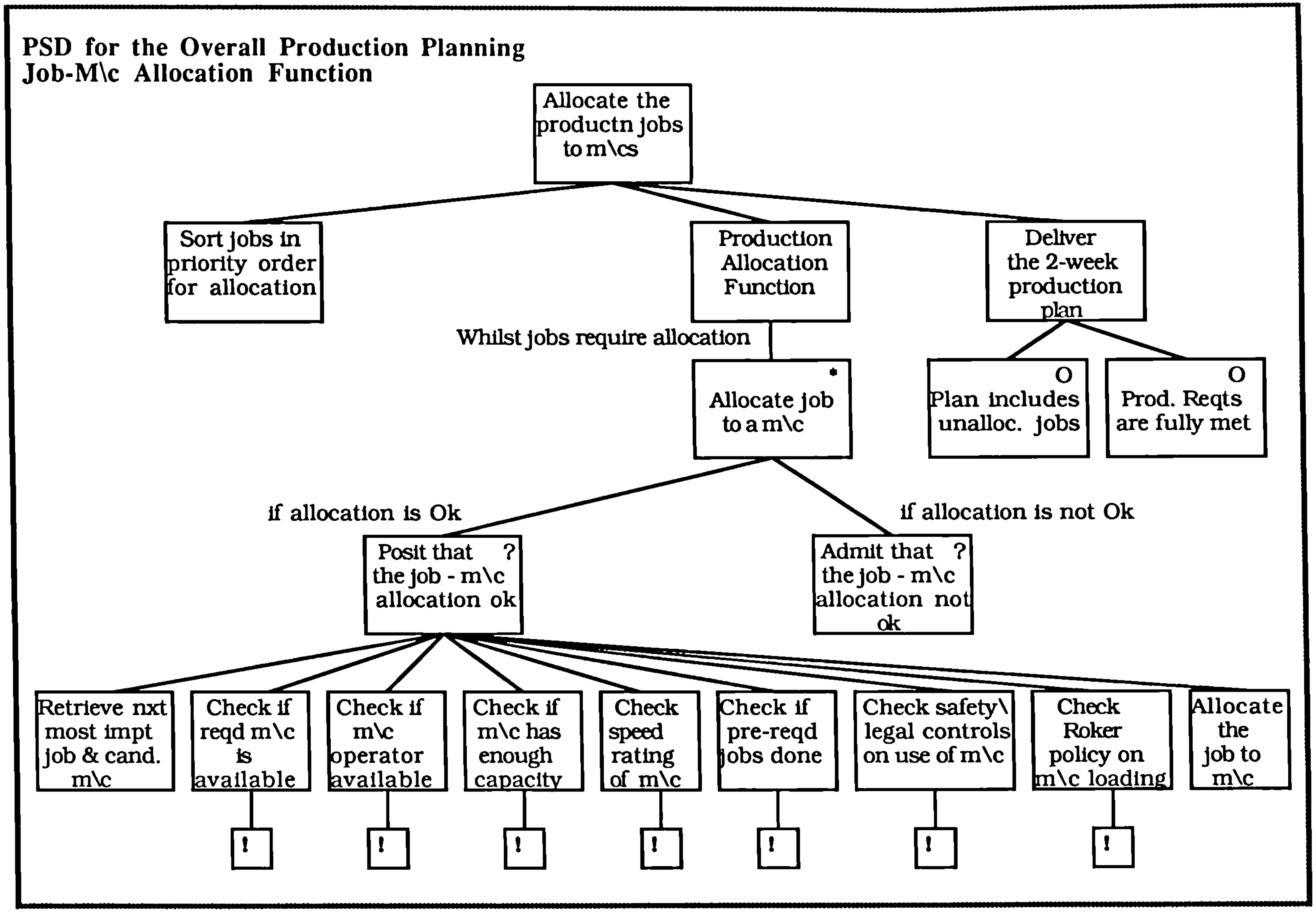
The allocation of jobs to machine takes place by job. In all cases all of the constraints on the job and the machine must be met. Priority jobs must be allocated to machines before all others. Roker priority jobs are :

- those jobs for which the necessary machines are regarded as rare, and
- those jobs for which required trained operator availability is stated as rare.

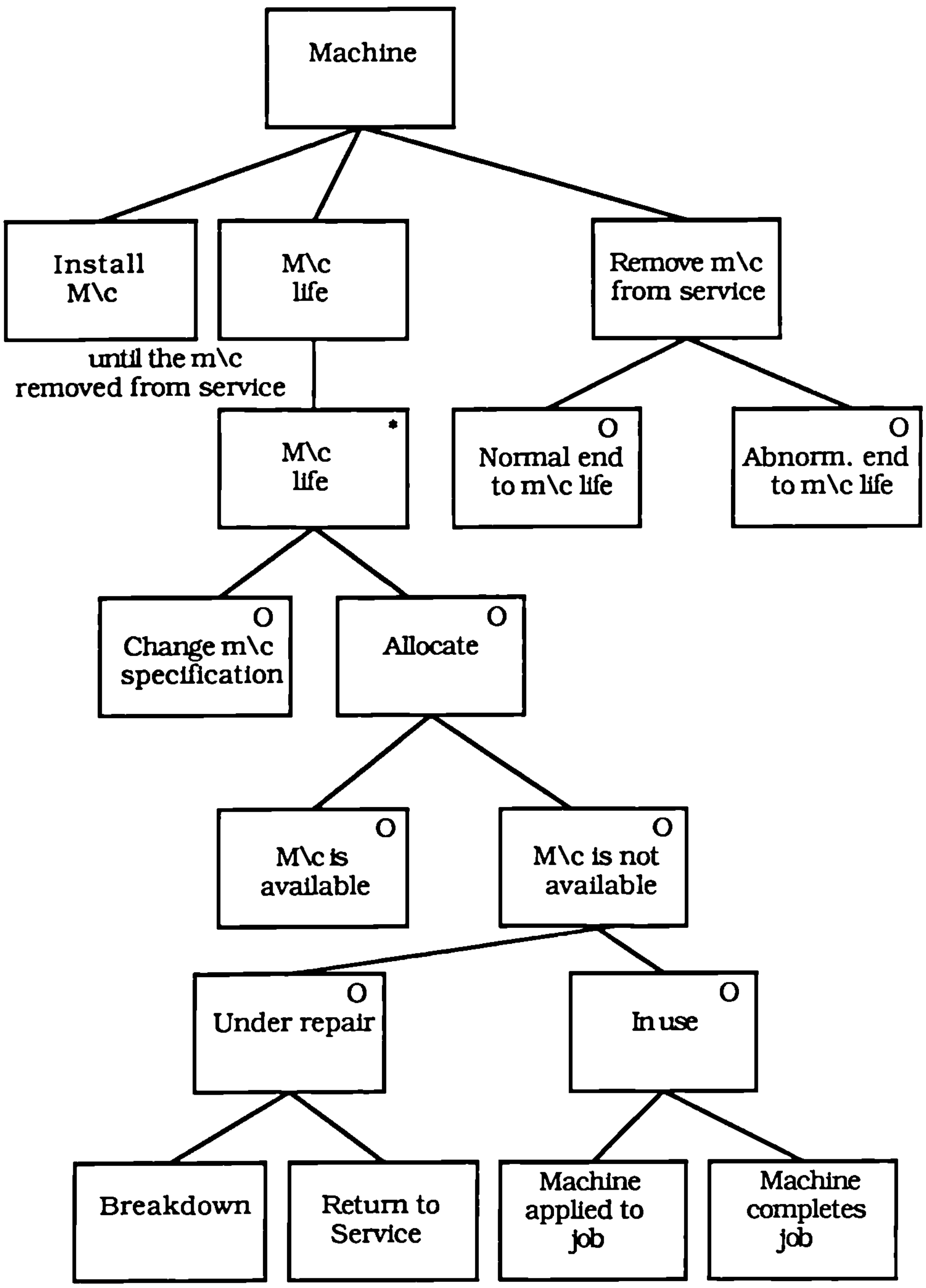
Therefore jobs meeting these two criteria should be selected and allocated to machines first. The order in which the constraints are applied to the proposed job-machine match is given in the accompanying function structure.

Outstanding machine idle time and unfulfilled production orders must then be reported to the Production Manager.

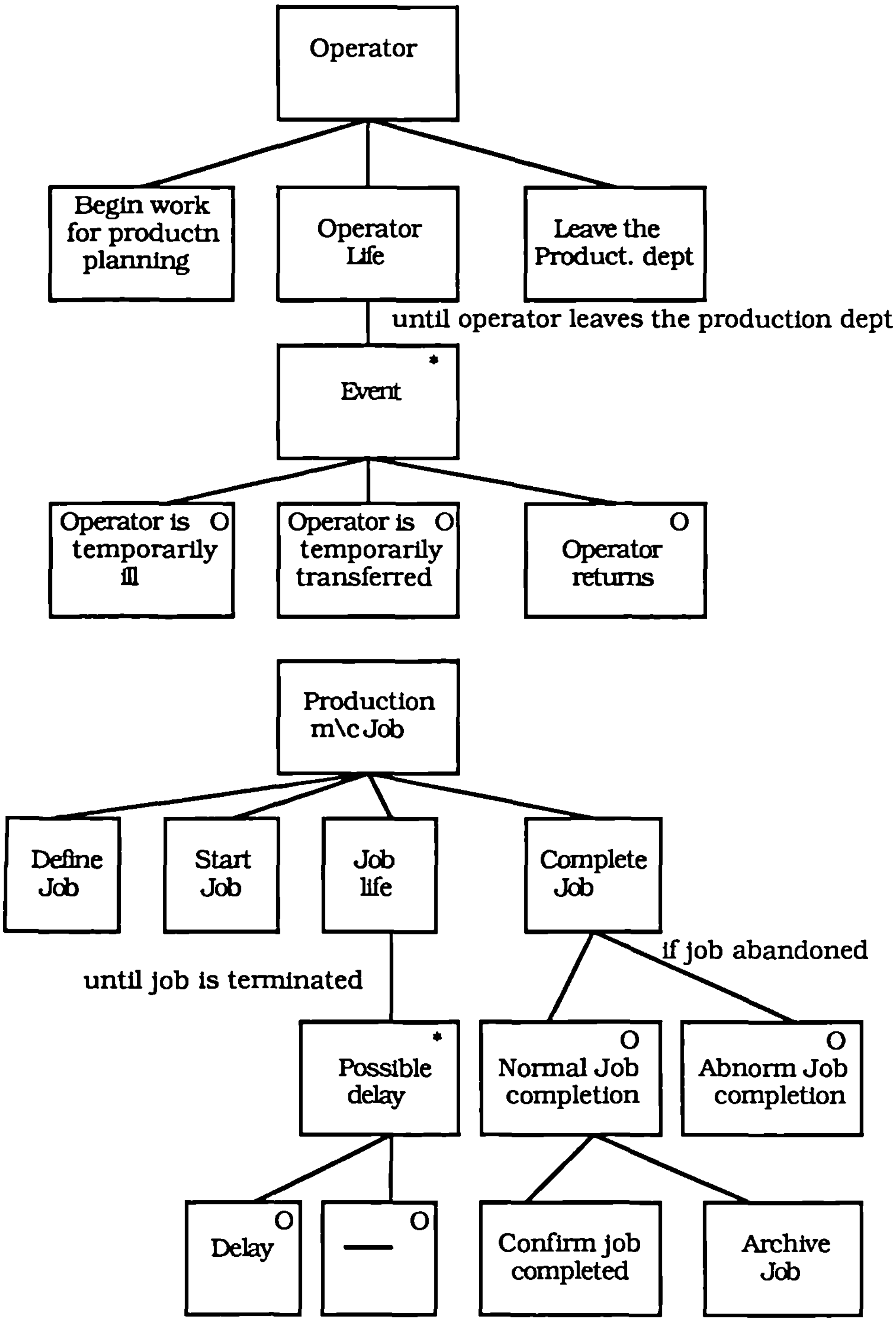
During reallocation of an existing function the production manager has the ability to 'LOCK' certain parts of the production plan, so that it is not changed during reallocation.


PSD for the Overall Production Planning
Job-M\c Allocation Function
Allocate the productn jobs to m\cs
Sort jobs in priority order for allocation
Production Allocation Function
Deliver the 2-week production plan
Whilst jobs require allocation
Allocate job to a m\c *
O Plan includes unalloc. jobs
O Prod. Reqts are fully met
if allocation is Ok
if allocation is not Ok
Posit that the job - m\c allocation ok ?
Admit that the job - m\c allocation not ok ?
Retrieve nxt most impt job & cand. m\c
Check if reqd m\c is available
Check if m\c operator available
Check if m\c has enough capacity
Check speed rating of m\c
Check if pre-reqd jobs done
Check safety\ legal controls on use of m\c
Check Roker policy on m\c loading
Allocate the job to m\c
!
!
!
!
!
!
!

**PSD for the Production Planning Enitity 'Machine'**



**PSD's for the Production Planning Entities Operator'and Production\M\c Job**



## Abstract Source Scheduling System Template

Overview of the Scheduling Function:

The goal of the general scheduling function is to allocate a limited number of Resources, each with a finite Capacity, to fulfil a pre-specified number of Tasks, either at or before a stated Time.

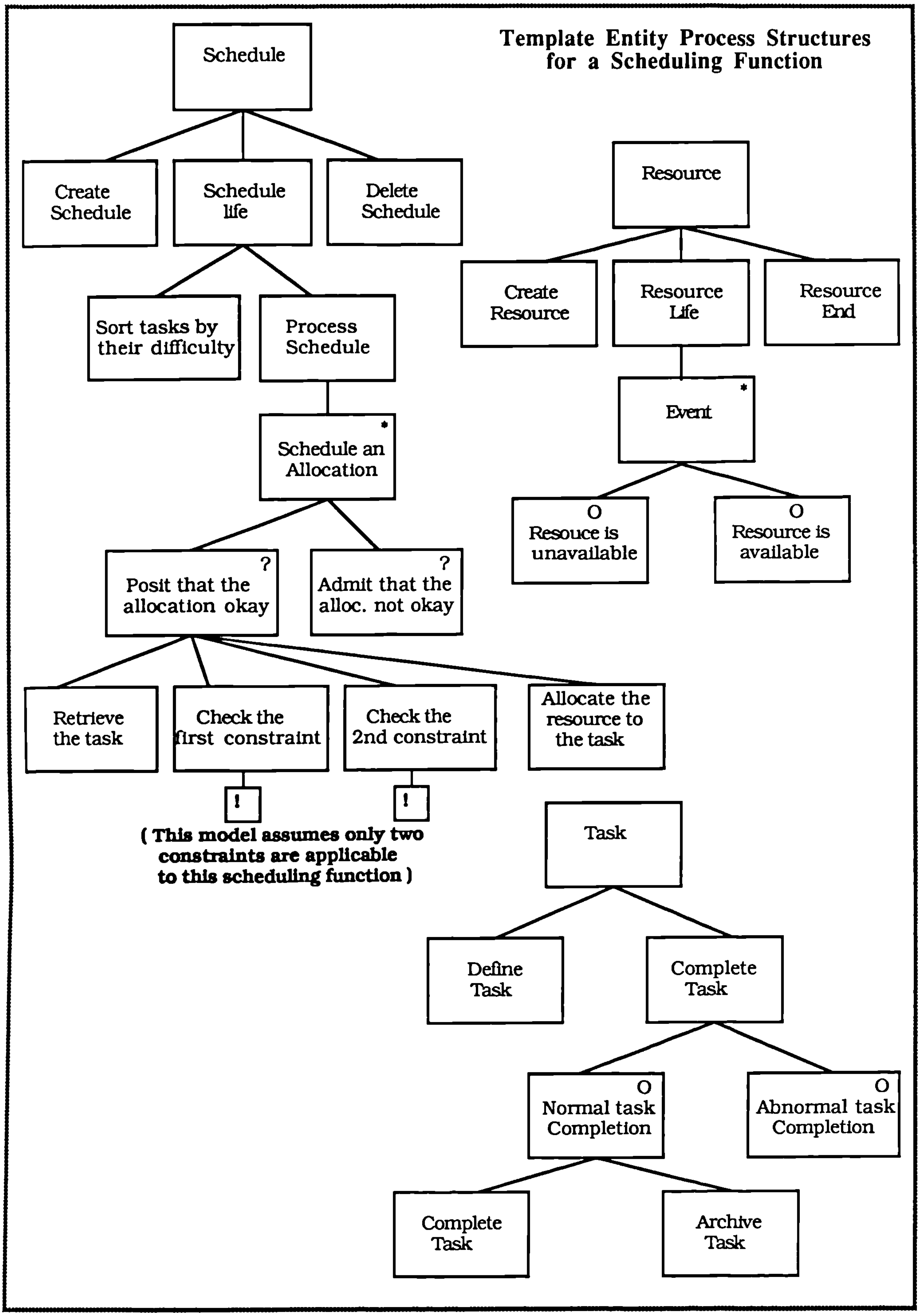
This allocation of Resources is controlled by one or many Constraints, which apply both to the Tasks to be fulfilled, and the Resources used to fulfil them. These constraints exist in 3 dimensions; the conceptual, spatial and temporal dimensions.

The detailed Scheduling function is controlled by the nature of and sequence that one or many of the Constraints are applied both to the outstanding Tasks to be completed, and the Resources available to complete them with. Those schedule allocations which are regarded as priority, either due to the scarcity of Resource or importance of the Task, are fulfilled first. This is achieved by sorting both the Tasks and the Resources.

The 3 major Entity Process Structures of a Scheduling function are :

- the Schedule function,
- the Resource entity, and, to a lesser extent,
- the Required Task entity.

The following page includes the general Entity Process Structures for these 3 structures.


Template Entity Process Structures
for a Scheduling Function
Schedule
Create Schedule
Schedule life
Delete Schedule
Sort tasks by their difficulty
Process Schedule
Schedule an Allocation *
Posit that the allocation okay ?
Admit that the alloc. not okay ?
Retrieve the task
Check the first constraint
Check the 2nd constraint
Allocate the resource to the task
!
!
( This model assumes only two constraints are applicable to this scheduling function )
Resource
Create Resource
Resource Life
Resource End
Event *
O Resouce is unavailable
O Resource is available
Task
Define Task
Complete Task
O Normal task Completion
O Abnormal task Completion
Complete Task
Archive Task

# Appendix D -Experimental Material for Empirical Studies 3&4, Chapter 4

# Sunderland Air Traffic Control System

## System Overview

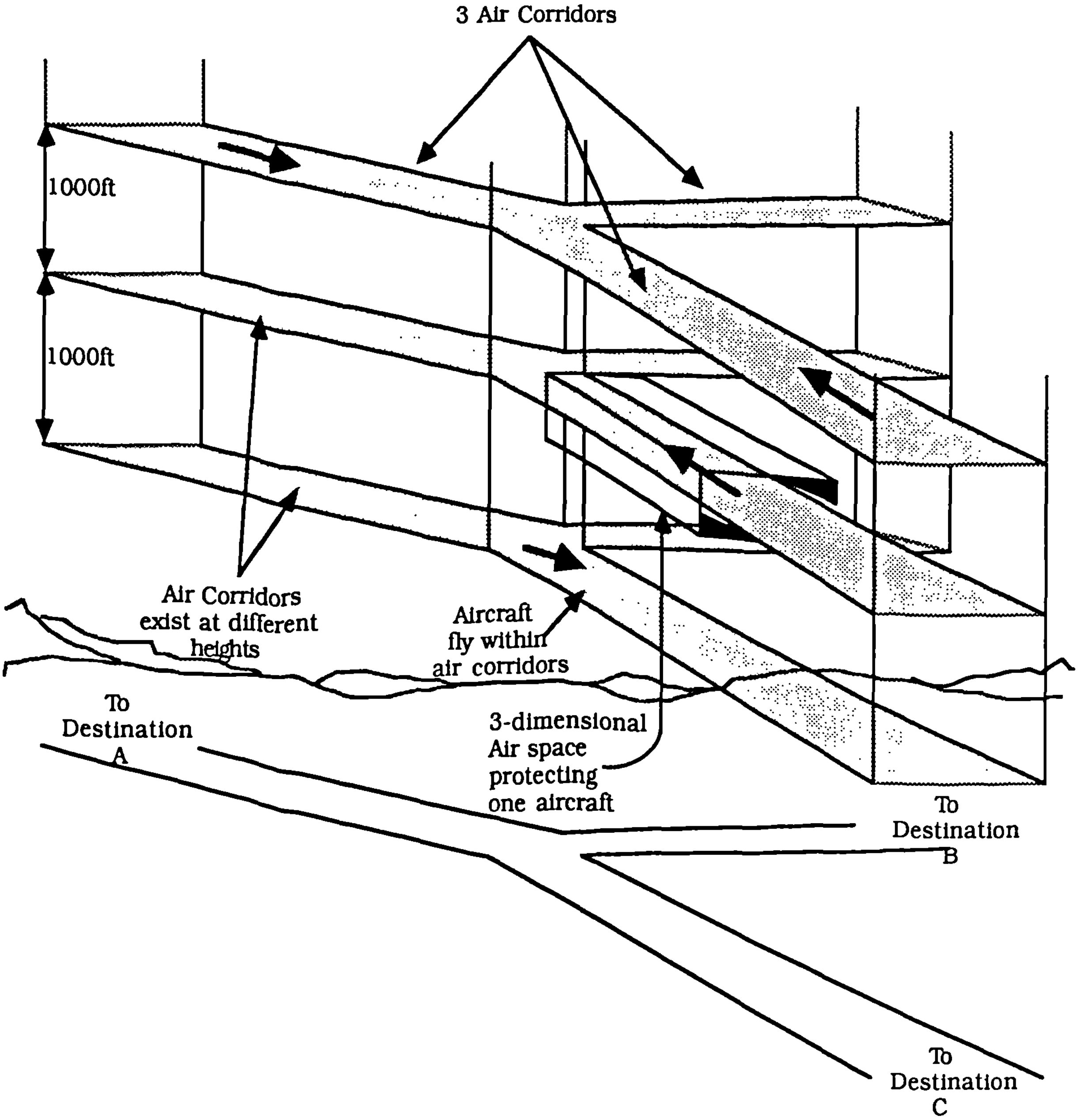
Sunderland Air Traffic Control Centre is implementing a computerised system to control commercial flights using Sunderland airport. This computer system is replacing a manual air traffic control system which can no longer cope with the increased traffic. The new system is being implemented in 2 phases:

i) development of a sub-system controlling airport traffic (i.e. landings, takeoffs and taxiing),
ii) development of a sub-system monitoring the current position of aircraft flying to and from Sunderland airport.

The first step has been completed and implemented: it will not be discussed any further. Your task is to focus on the second phase: developing a sub-system to monitor aircraft flying to and from Sunderland airport.

## The Existing Sub-system

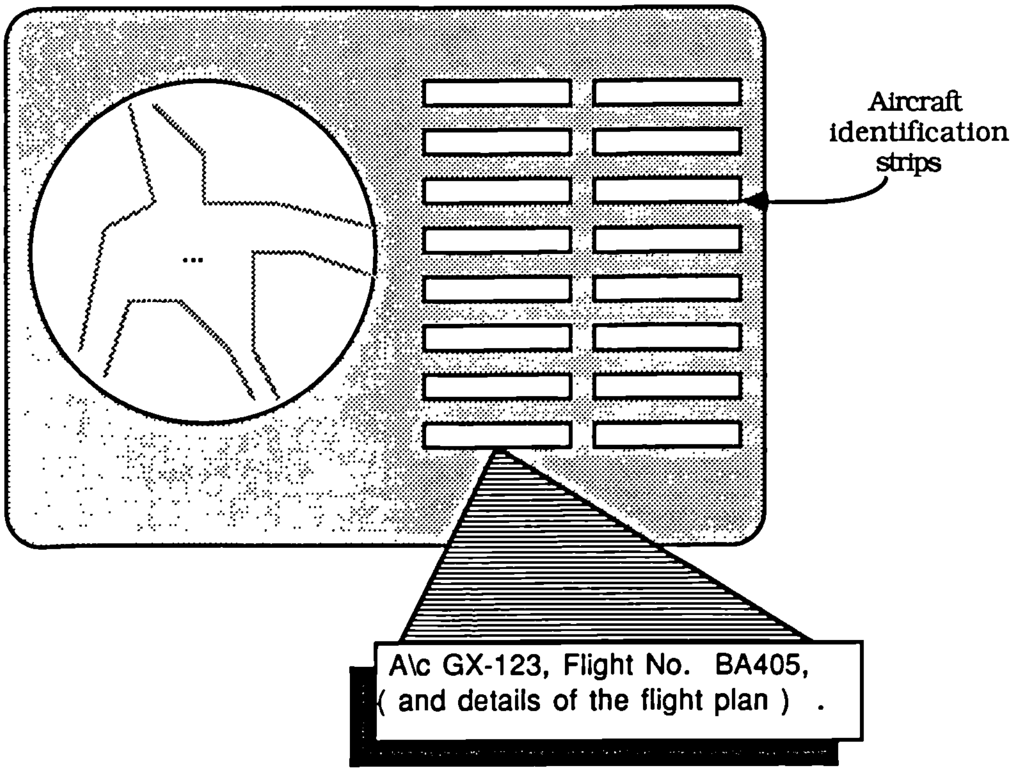
The sky around Sunderland airport is structured to improve the control of aircraft movements. Aircraft fly along unidirectional air corridors, to the airport and other destinations. Aircraft fly at different heights along these air corridors, so that many aircraft can use one air corridor at any one time: these heights are predetermined and carefully controlled, and usually exist at intervals of 1000 feet. Aircraft follow each other along these air corridors, separated by a distance presently determined by the type, size and speed of each aircraft. Several air corridors exist within a geographical area; an example of one such geographical area containing three air corridors is given in figure 1.




Simplified 3-dimensional model of the airways within one geographical area: aircraft fly at different heights along air corridors.

Figure 1

The air traffic controller directs aircraft to fly at certain heights within air corridors, from decisions based on the data available to him: current aircraft positions are indicated from radar signals displayed on the air traffic controller's radar screen. The air traffic controller also records important data about each aircraft (e.g. flight number, details of flight plan, instructions to pilot, etc.) on cardboard strips, kept next to the radar screen. The air traffic control console used in the existing system is given in figure 2. To control aircraft movements the air traffic controller issues verbal commands to aircraft pilots, who repeat the command for confirmation before directing the aircraft accordingly. An air traffic controller controls several flights within a single geographic area: several air traffic controllers control all aircraft in one geographic area.




Existing Air Traffic Control Console

Figure 2

The air traffic controller must instruct the pilot to ensure the safety and timely arrival of each aircraft. Each flight of an aircraft is guided by a flight plan, which has been determined jointly beforehand by air traffic control and the airline. The air traffic controller must ensure an aircraft follows the plan as closely as possible, by issuing commands to the pilot to change course at the appropriate times. An example flight plan is given in figure 3.

| Time | Air corridor | Height |
|---|---|---|
| 16:04 | 1-23 | 10,000ft |
| 16:19 | 1-23 | 11,000ft |
| 16:34 | 1-19 | 15,000ft |
| | etc | |

Individual flight steps in the flight plan.

Example Flight Plan
for one aircraft on
one flight.

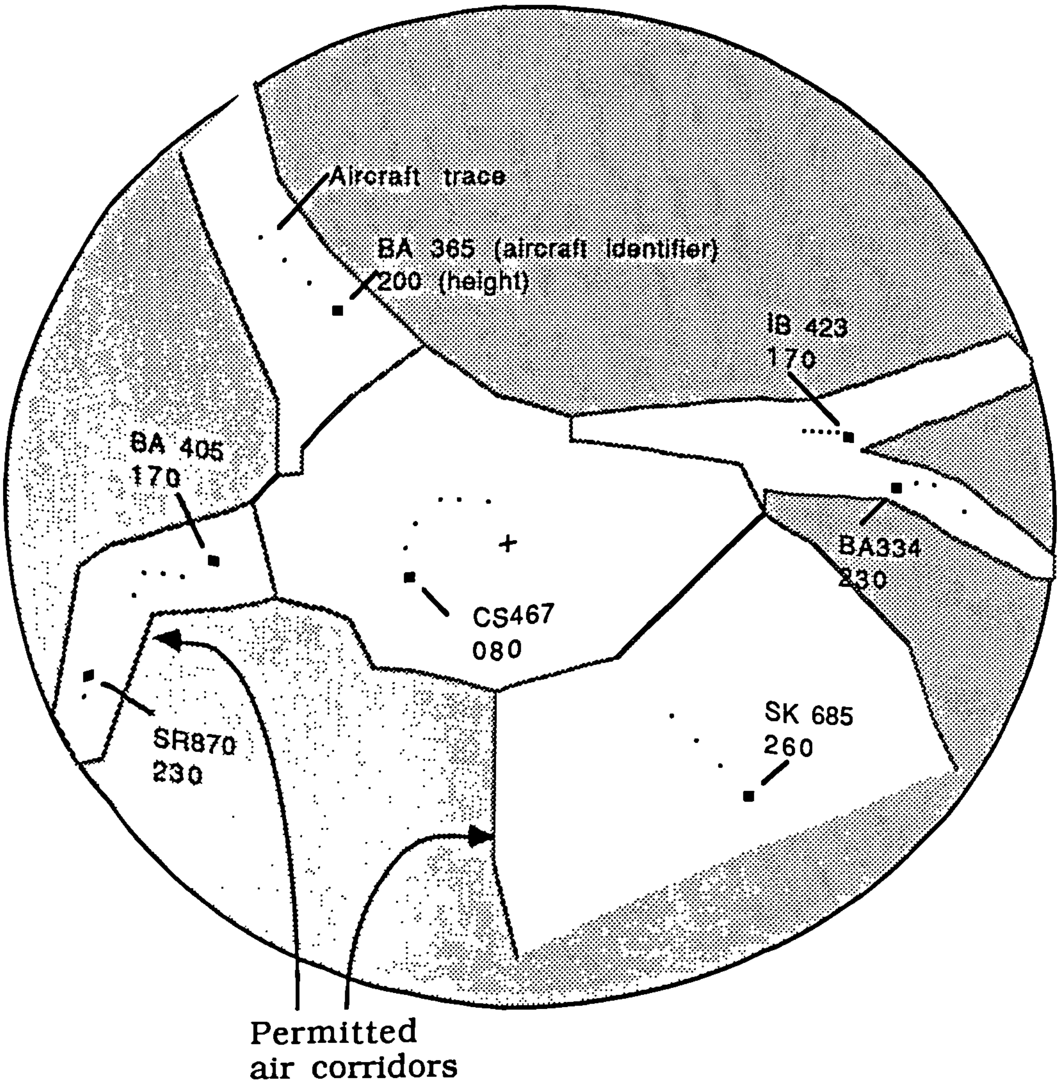
Figure 3

The air traffic controller must keep aircraft sufficiently far apart to minimise the danger of midair collisions: this is achieved by redirecting aircraft to different air corridors or heights, whenever two or more aircraft are observed to be too close together.

**The Required Sub-system**

The new computerised system must support existing system features and automate some of the manual processes originally undertaken by the air traffic controller. The sub-system has two major objectives: (i) avoid collisions between aircraft approaching and leaving Sunderland Airport, (ii) ensure aircraft reach their destination on time. Required features of the system are described in turn.

As with the existing system the new system must be able to interpret incoming radar signals from aircraft. It should also produce the computerised radar screen described in figure 4: data originally recorded on cardboard strips is now displayed in computerised form on the radar screen.




The required air traffic controller screen for each air traffic controller (screen shows all aircraft in geographical area, not just aircraft under an individual's control ).

Figure 4

To ensure aircraft safety the system must alert the air traffic controller whenever two aircraft come too close. Aircraft are surrounded and protected by an air space which no other aircraft is legally permitted to enter. This air space is a three-dimensional area which exists within a given air corridor and height (for example, see figure 1). The dimensions of this air space are dependent upon aircraft type, speed, height, elevation and direction. Aircraft identity, position and height are determined from radar signals whereas its speed and direction must be calculated from the aircraft's trace (the previous series of aircraft positions, see figure 3). All aircraft's air spaces should be recalculated and monitored whenever data updating aircrafts' positions become available. When the air spaces of two aircraft overlap the air traffic controller must be informed as a matter of urgency.

The system must also monitor each aircraft to ensure it does not deviate from either the air corridor or the flight plan. Each flight plan is divided into a number of flight steps, which are given by the air traffic controller to direct the aircraft to use given air corridors at certain times during the flight. Whenever the aircraft strays from the required path the system must inform the air traffic controller immediately.

Aircraft monitoring can result in unexpected changes to an aircraft's direction, which must be reflected in the system; the air traffic controller must be able to update the flight plan accordingly whenever such a change is made.

**Your Instructions**

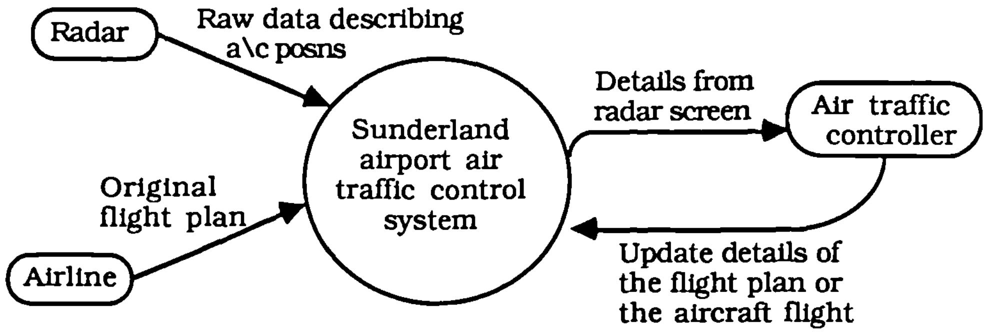
You are required to develop a context and level-0 DFD to represent the required logical system described in this document. If necessary describe some of the processes of your level-0 DFD with a narrative description.

**Scope of the Required Sub-system**

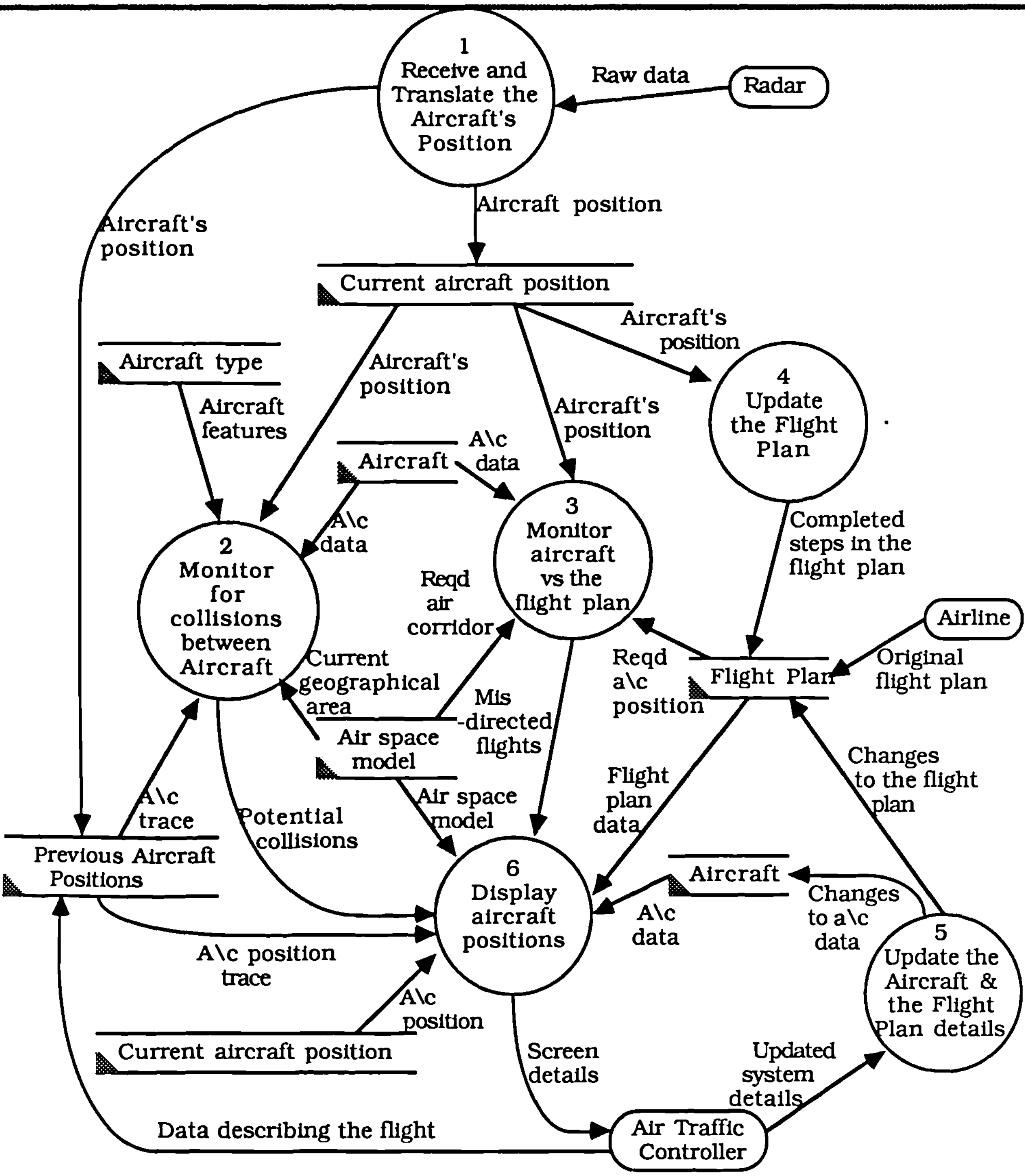
The following are beyond the scope of the sub-system that you should develop:

(i) airport control, including the management of aircraft during taxiing, landing and takeoff,
(ii) aircraft flying over Sunderland airspace, without landing at Sunderland airport,
(iii) communications between the pilot and air traffic controller (all input to the system from the pilot enters through the air traffic controller),
(iv) emergency flights requiring special clearance and priority,
(v) the effects of severe weather, and other exceptional conditions which might require aircraft to take unexpected actions.

# Solution to Sunderland's Air Traffic Control Problem




*Context DFD for Sunderland Airport ATC System*




*Level-0 DFD for Sunderland ATC System*

# Study 3&4 Checklist

List of components checked for completeness in subjects' solutions. A list of error types looked for in subjects' solutions is also included.

| Completeness Checks | | | |
|---|---|---|---|
| **Context DFD** | | P1 - Previous A\c Positions | |
| Process - Sunderland ATC | | P1 - Current A\c Positions | |
| Source - Radar | | Current A\c Position - P2 | |
| Source - Airline | | Current A\c Position - P3 | |
| Source - Air Traffic Controller | | Current A\c Position - P4 | |
| Input - Raw data for radar | | Aircraft Type - P2 | |
| Input - Original flight plan | | Previous A\c Position - P2 | |
| Input - Update details from the ATC | | Air Space Model - P2 | |
| Output - Details from radar screen | | Aircraft - P2 | |
| **Level-0 DFD** | | P2 - P6 | |
| Process 1 - Receive aircraft position | | Aircraft - P3 | |
| Process 2 - Monitor for collisions | | Flight Plan - P3 | |
| Process 3 - Monitor against flight plan | | Air Space Model - P3 | |
| Process 4 - Update flight plan | | P3 - P6 | |
| Process 5 - Update a\c and flight plan details | | P4 - Flight Plan | |
| Process 6 - Display Radar Screen | | P5 - Flight Plan | |
| Source - Radar | | Air Space Model - P6 | |
| Source - Airline | | Aircraft - P6 | |
| Source - Air Traffic Controller | | Flight Plan - P6 | |
| Input - Raw data for radar | | Previous A\c Positions - P6 | |
| Input - Original flight plan | | Current A\c Positions - P6 | |
| Input - Update details from the ATC | | | |
| Input - ATC - Previous Aircraft Positions | | | |
| Output - Details from radar screen | | | |

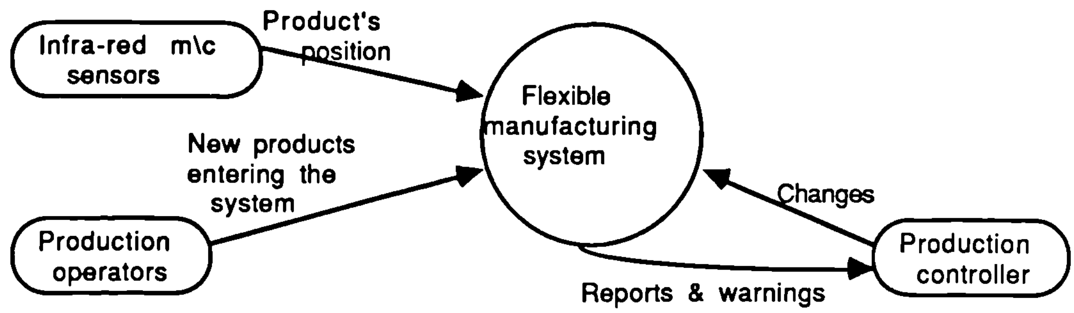
| Errors Checks | | | |
|---|---|---|---|
| *Syntax* | | *Semantic* | |
| Confuse data stores and sources on Ctxt DFD | | Failure to identify major processes | |
| Missing data flow labels\arrow (>4) | | Determine wrong data stores for processes | |
| Misuse of other data flow DFD symbols | | Failure to recalculate air space | |
| Breaking DFD rules | | Combine P4 & P5 unnecessarily | |
| Failure to draw both DFDs | | Failure to recognise the problem scope | |
| | | Too many data stores for processes | |
| | | Source rather than target objects in spec | |

# Description of Flexible Manufacturing System

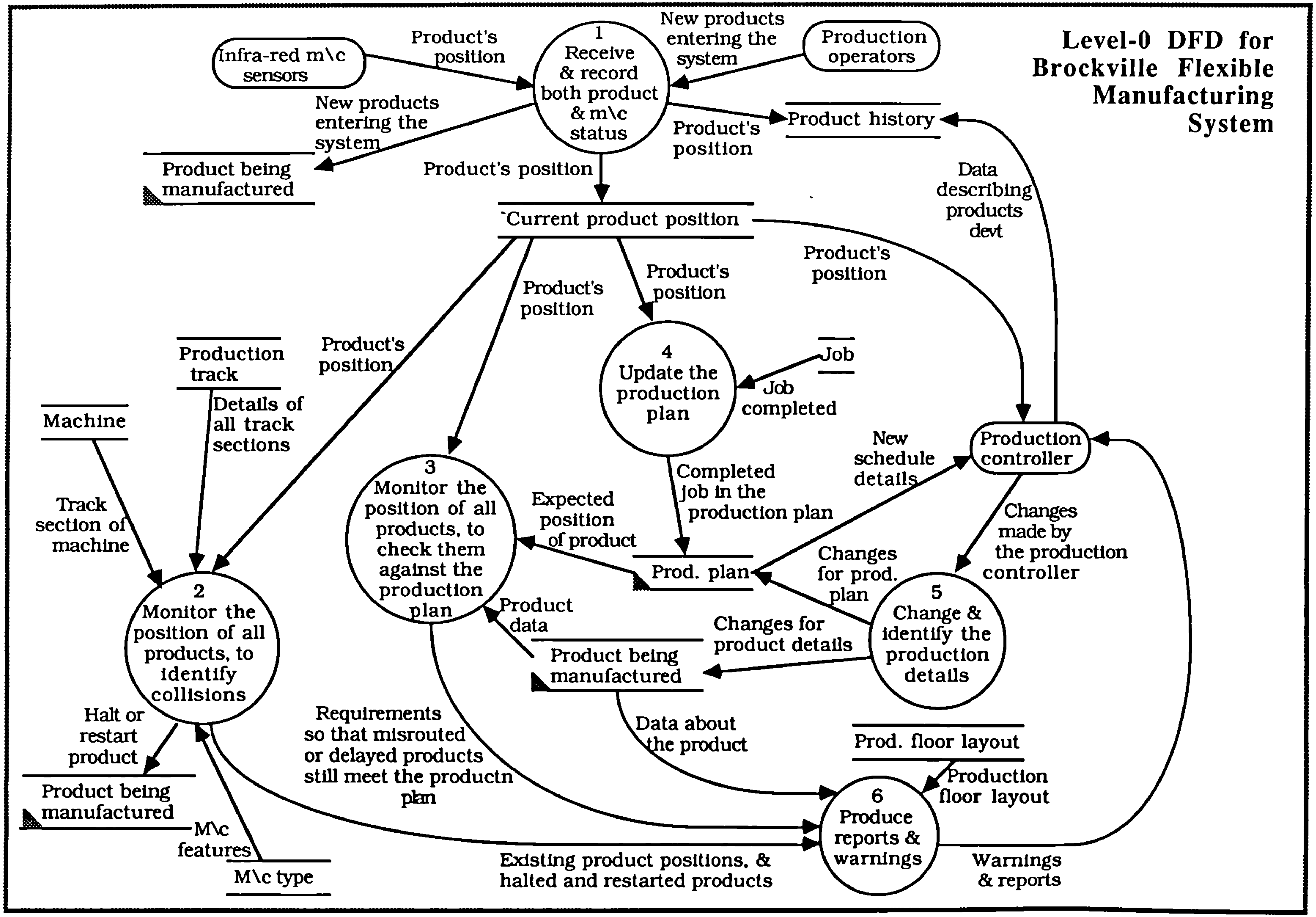
<u>Brockville Flexible Manufacturing System</u>

Brockville Precision Tools is a high-tech company manufacturing products (precision tools) using the latest computerised production techniques. The company is moving towards full automation of production facilities, in order to keep human operator intervention to a minimum.

Recently a new system was installed, to monitor production. The system identifies delays and potential accidents during production, so that the automated handling system can take appropriate action. The production monitoring system is described in the 2 accompanying data flow diagrams, and in the supporting narrative.




Context DFD for Brockville Flexible Manufacturing System


Level-0 DFD for Brockville Flexible Manufacturing System
Infra-red m\c sensors
Product's position
1 Receive & record both product & m\c status
New products entering the system
Production operators
New products entering the system
Product being manufactured
Product's position
Product history
Product's position
Current product position
Data describing products devt
Product's position
Product's position
Product's position
Product's position
4 Update the production plan
Job
Job completed
Production track
Details of all track sections
Machine
Track section of machine
3 Monitor the position of all products, to check them against the production plan
Expected position of product
Completed job in the production plan
New schedule details
Production controller
Changes made by the production controller
Prod. plan
Changes for prod. plan
5 Change & identify the production details
Product data
Product being manufactured
Changes for product details
2 Monitor the position of all products, to identify collisions
Halt or restart product
Product being manufactured
M\c features
M\c type
Requirements so that misrouted or delayed products still meet the productn plan
Data about the product
Prod. floor layout
Production floor layout
6 Produce reports & warnings
Existing product positions, & halted and restarted products
Warnings & reports

This narrative explains 3 processes of the level-0 DFD in more detail.

Process 2 - Monitor the position of all products, to avoid collisions

The aim of this process is to ensure no two products being manufactured come together during the manufacturing process.

During production products are passed along lines of manufacturing machines by a complicated series of conveyor belts and automatic handlers. Each individual line of machines is called a track, and each track is divided into many sections, which can only legally contain one product at any time. Product positions are determined by infra-red sensors laid along the tracks.

This process invesitgates the current position of all products to ensure no track section contains more than one product. If two products are in the same section one product is halted automatically, and restarted again once the other product has cleared that section. The production controller is warned of any potential accidents, so that he may reroute products.

Process 3 - Monitor the position of all products, to check the production plan is met

A production plan determines the order of machines which a product must follow during manufacture. This process checks to ensure that the tracks followed by a product are those intended, by comparing the current product position with that given in the production plan. Diversions of any sort from the plan are reported to the production controller.

Process 4 - Update the production plan

When sensors detect an individual manufacturing job using one machine has been completed on a product the production plan for that product is updated to indicate a further step in the plan has been fulfilled.

## Subject Experience in Study 4

| Subject | Analytic experience | Programming experience |
|---|---|---|
| E1 | 15-yrs local govt\financial | 12-yrs CICS COBOL, RPGII |
| E3 | 20-yrs local govt\civil service | 20-yrs COBOL, GMAP |
| E4 | 10-yrs with local govt | 18-yrs COBOL |
| E5 | 6-months with local govt | 6-yrs COBOL |
| E7 | 5-yrs local govt, & SSADM | 3.5-yrs FORTRAN, FORTH |
| E8 | 7-yrs hands on with SSA | Unknown |
| E9 | 3-yrs hands-on with CASE | Little |
| E10 | 2-yrs teaching SSA techniques | 12-yrs LISP, PROLOG, etc |
| E11 | 7-yrs hands-on with SSA | 7-yrs COBOL |
| E12 | 15-yrs hands-on, 4 of SSA | 6-yrs COBOL & PASCAL |

Previous pogramming and analysis experience of all subjects

## Example Protocol Transcript for Studies 3&4

| Analytic Strategy | Mental Category | Physical Behaviour | Mental Behaviour |
|---|---|---|---|
| *Gather information* | Generate General Plan | | I'm going to read the reusable DFD. |
| | | *Read the reusable DFD for 15 seconds* | |
| *Reuse* | Assertion | | So the FMS has a process to monitor products for collisions, |
| | Generate Source Hypothesis | | there is a process to monitor collisions, |
| | Test Source Hypothesis | | so how is that analogous ? |
| | Extend Source Hypothesis | | products pass along lines of machines, |
| | Test Source Hypothesis | | er, |
| | Extend Source Hypothesis | | these are sequential lines of machines, |
| | Extend Analogy Hypothesis | | machine lines are analogous to air corridors, |
| | Test Analogy Hypothesis | | and aircraft are analogous to, um ? |
| | Extend Analogy Hypothesis | | aircraft are analogous to products, |
| | Extend Target Hypothesis | | so the monitor process becomes monitor products for collisions, |
| | Test Target Hypothesis | | er, yeh, that's correct, |
| | Extend Target Hypothesis | | so monitor products to avoid collisions among aircraft. |
| | | *Draw the process on target solution* | |
| *Construct* | Generate General Plan | | Let's add the data stores from this process, |
| | Extend General Plan | | so look at the air traffic control requirements again. |
| | | *Read problem requirements document briefly* | |
| | Generate Target Hypothesis | | We need the aircraft details to calculate the air space, |
| | Extend Target Hypothesis | | so we need a data store access to 'aircraft', |
| | Test Target Hypothesis | | yep, that's right. |
| | | *Draw the data store and flow on target solution* | |
| | Generate Target Hypothesis | | Air Corridor tends to provide important information for this process, |
| | Extend Target Hypothesis | | it does seem to be relevant somehow. |

| Analytic Strategy | Mental Category | Physical Behaviour | Mental Behaviour |
|---|---|---|---|
| | Generate General Plan | | So let's run through what I've got so far. |
| *Summarise solution* | | *Look at the solution DFD* | |
| | Model-based Testing | | We have a radar, |
| | Model-based Testing | | which input the position of aircraft, |
| | Model-based Testing | | this records the position and does something with it, |
| | Model-based Testing | | then it passes this to the monitor process |
| | Model-based Testing | | to check for collisions, |
| | Model-based Testing | | which then accesses the data stores, |
| | Model-based Testing | | ummmm.... |
| *Evaluate against the target* | | *Read the problem requirements document* | |
| | Generate Target Hypothesis | | Calculation of the air space needs data about the aircraft type, |
| | Extend Target Hypothesis | | so we need to have aircraft type in this process, |
| | Other | | so this is something which is missing, |
| | Extend Target Hypothesis | | aircraft type receives data from the process, |
| | | *Draw the data store and flow on the solution* | |
| | Test Target Hypothesis | | er, no, that's not right !! |
| | Modify Target Hypothesis | | the data flow should be the other way around, |
| | Extend Target Hypothesis | | so data flow from the data store. |
| | | *Redraw the data flow* | |
| *Evaluate against the analogy* | Generare General Plan | | Let's check that against the analogy. |
| | | *Read the reusable DFD* | |
| | Generate Analogy Hypothesis | | Aircraft type is equivalent to product type, |
| | Extend Analogy Hypothesis | | however there is no equivalent on the FMS, |
| | Extend Analogy Hypothesis, | | we have a machine type which can be mapped, |
| | Test Analogy Hypothesis, | | I think, |
| | Test Analogy Hypothesis, | | yeah, okay, |
| | Extend Analogy Hypothesis, | | lets say that aircraft type maps to machine type. |
| | Generate General Plan | | So what else have I missed from the analogy ? |
| | Generate Analogy Hypothesis | | there is no mapping to this data flow, |
| | ..... | | so let me think.... |

# Appendix E - Paper-Based Evaluation of the Problem Identifier Module

## Production Planning in the Roker Manufacturing Company

Roker Manufacturing is a medium-sized company making heavy equipment for shipbuilding. Their plant consists of several workshops, each containing specialised machines for the construction of different types of equipment. Production is planned monthly by allocating manufacturing jobs to appropriate machines in the workshops. This document focuses on the monthly production planning process. You are required to analyse this process and model it using techniques described in the accompanying document. Do not consider other aspects of the Roker manufacturing system.

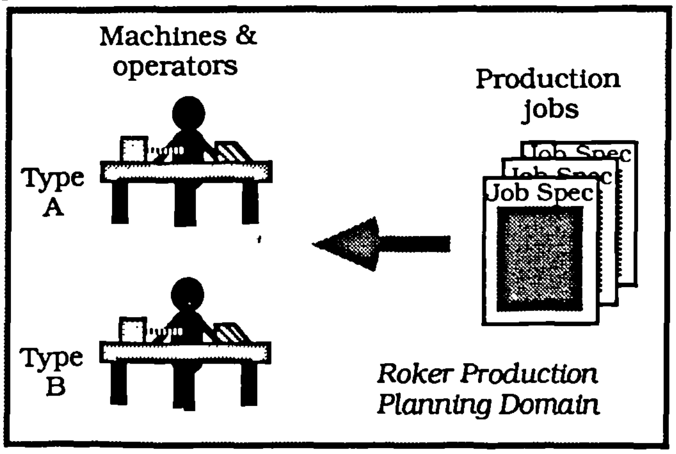
The production planning process is carried out at the beginning of every month by a computerised scheduling system. The aim of this system is to allocate the monthly quota of manufacturing jobs to machines in a way that:

* maximises the use of machines,
* ensures that a maximum number of jobs are completed by their deadline.

The allocation of a job to a machine is constrained by several important limitations, including:
* the suitability of a machine to manufacture a product,
* the availability of skilled operators to use machines,
* the sequence in which manufacturing jobs must be carried out,
* the ability of certain machines to complete a job in the time allowed.

In addition, certain jobs are given priority allocation to machines. Priority jobs occur when:
* the necessary machines are rare,
* the necessary skilled operators are rare.

In this study you are required to describe facts about the production planning domain by following instructions given in the Ira Toolkit. To enter Ira use the instructions given on the accompanying sheet.



# Help Document

This document is intended to help you develop a simple description of Roker's Production Planning domain. The model is developed by following instructions in the seven steps below. Your description of the Production Planning domain should be written on the accompanying Answer Sheet. Each step in the instructions is accompanied by an example of a Personnel System in which the arrival of staff in an organisation is recorded.

## *Step 1 - Identify the Most Important System Goal*

Steps 1 & 2 encourage you to identify some background information about the Production Planning domain. Firstly identify the most important goal of the system. Use your own terminology to describe the goal, and add it to Step 1 on the Answer Sheet.

*Example*
The major purpose of the Personnel System is to record data about staff joining and leaving the organisation, so the goal is:

*'Record data about Staff arrivals'*.

## *Step 2 - Identify the Main System Functions*

Select up to four functions which best represent Roker's computerised Production Planning system. Focus on functions which support the major system goal identified in Step 1 and ignore functions which only occur in exceptional circumstances. Underline your selected functions in Step 2 on the Answer Sheet. Note that many systems will have fewer than four major functions.

*Example*
The Personnel system *records* the movement of staff to and from the organisation, so the most appropriate function in this system is *Record*.

## *Step 3 - Identify the Domain Entities*

This step identifies the major entities in the Production Planning domain. You may identify up to four domain entities. This can be achieved in two ways:

* entities may be physical objects, so identify physical entities in the Production Planning domain. Prefer entities which are linked to the major goal identified in Step 1,

* entities are related to functions. Each function processes or does something to an entity. Identify entities which are directly processed by the functions underlined in Step 2.

Add these entities to the entity list in Step 3 on the Answer Sheet.

*Example:*
In the Personnel domain *Outside-world* and *Organisation* are physical entities relevant to the system goal. In addition, the *Record* function acts upon the Staff entity, since it records staff joining the organisation, so Staff is another candidate entity. This suggests that the personnel system domain has three important entities:
*Staff, Organisation & Outside-world.*

## *Step 4 - Identify the Domain Structure*

The three parts of Step 4 identify the structure of the Production Planning domain by specifying relationships between entities identified in Step 3. How to identify these relationships is described below.

### *Step 4(a) - Specify the Structure of the Domain*

This step specifies the structure of entities identified during the previous step. You may want to sketch the Production Planning domain in a similar way to the Personnel domain example in Figure 4. This step consists of three mini-steps. If necessary you should go back and change facts about the Production Planning domain identified during Steps 1-3.

i) Consider each function underlined in Step 2. Each function processes one major entity, so select that entity from the list of entities in Step 3 of the Answer Sheet, and add it twice to the Entity-2 column of Step 4 on the Answer Sheet. In the Personnel example the computer system *Records Staff* movements in and out of the Organisation, so the *Record* function processes *Staff*.. The staff entity was added twice to the list, see Figure 1.

| Entity-1 | Entity-2 | Relation |
|---|---|---|
| | *Staff* | |
| | *Staff* | |

Figure 1

ii) For each function there is an initial position and a final position for the entity processed by the function. You should identify the initial and final position of the entity processed by each function. Give the starting and final entities for each entity processed by each function and add them to the relevant Entity-1 columns in Step 4 in the Answer Sheet. The resulting list should identify the starting and final entities for each main entity, see Figure 2.

| Entity-1 | Entity-2 | Relation |
|---|---|---|
| *Organistn* | *Staff* | |
| *OutsideWd* | *Staff* | |

Figure 2

iii) The relationship between each entity processed by a function and the starting and final positions of that entity can be specified in more detail. At any time the starting and final positions for each entity may contain one or many entities, so for each pair of entities in the list in Step 4 use one of the following relationships between entities to describe their starting and final positions, and add these entity-relations to the final column in step 4:
   * A contains-one B (B contains one entity A),
   * A contains-many B (B contains many As),

   where B is the entity processed by the function and A represents the initial and final positions of the entity. In the Personnel domain many staff can either be in the Outside-world or in the Organisation, so see Figure 3.

| Entity-1 | Entity-2 | Relation |
|---|---|---|
| *Organistn* | *Staff* | *contains-many* |
| *OutsideWd* | *Staff* | *contains-many* |

Figure 3

The final version of the list in Step 4(a) is shown in Figure 4.

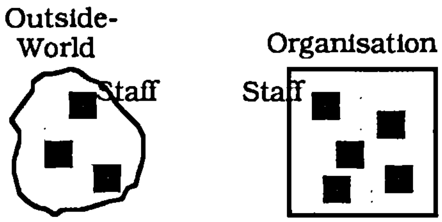



Figure 4

## *Step 4(b) - Number of Entities in the Domain*

Your model of the production planning domain can be further developed by stating how many times entities exist in the production planning domain. Each entity may exist one or many times in the Production Planning domain. For each different entity in the Entity-1 column state either that:

* World has-one 'entity' (there is only one 'entity' in the World), *or*
* World has-many 'entity' (there are many 'entities' in the World).

Add each entity and the appropriate relation to the relevant lines in Step 4 on the Answer Sheet.

*Example*
In the Personnel domain the Organisation and Outside-world entities only occur once, so two more facts about the domain were identified, see Figure 5.

| Entity-1 | Entity-2 | Relation |
|---|---|---|
| World | *Organistn* | *has-many* |
| World | *OutsideWd* | *has-many* |

Figure 5

This can be represented graphically in Figure 6:

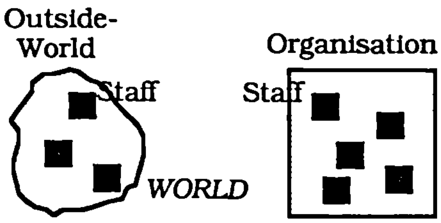



Figure 6

## *Step 4(c) - Complete the Structure of the Domain*

You may want to identify other relations between entities which were not identified during Steps 4(a) and 4(b). These features of the domain are best recognised by sketching the Production Planning domain, if you have not already done so. Use the following two relations to describe any additional features of the Production Planning domain:

* A has-one B (there is one B in A),
* A has-many B (there are many B in A),

Add each fact (entity-relation-entity) to the list in Step 4 on the Answer Sheet in a similar way to facts identified in Steps 4(a) and 4(b).

*Example*
The final version of the list for Step 4 in Figure 7 is:

| Entity-1 | Entity-2 | Relation |
|---|---|---|
| *Organistn* | *Staff* | *contains-many* |
| *OutsideWd* | *Staff* | *contains-many* |
| *World* | *Organistn* | *has-many* |
| *World* | *Outside Wd* | *has-many* |

Figure 7

## *Step 5(a) - Specify Functions in terms of Entities*

This step formally describes the functions identified in Step 2 in terms of entities and structures identified in Steps 3 & 4. List each function selected during Step 2 in the left-hand column of Step 5 in the Answer Sheet, then for each function you should identify:

* the entity processed by that function,
* initial position of the entity,
* final position of the entity.

Add these entities to the second,third and fourth columns in Step 5 on the Answer Sheet. The source and destination entities should correspond to initial and starting positions of main entities identified in the previous Step.

You may find it useful to sketch the functions as demonstrated by the Personnel example, see Figure 9.

*Example*
The list of function descriptions for the Personnel domain is given in Figure 8, and these functions are represented graphically in Figure 9:

| Function | Entity | Source | Destntn | Move-one |
|---|---|---|---|---|
| *Record* | *Staff* | *Outsde W* | *Organstn.* | |

Figure 8

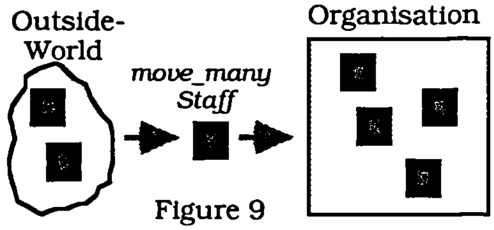



Figure 9

## *Step 5(b) - Detail the Functions*

Function definitions can be specified in more detail by identifying each function as either:
* a single movement (move-one), so that only one entity can be moved at any time, or
* a multiple movement (move-many), so that many entities can be moved at any time.

Add these movements to the relevant right-hand columns of each function in Step 5 on the Answer Sheet.

*Example*
In the personnel domain many staff may arrive in the organisation at any time, so:
* <u>move-many</u> *staff* from the *outside world* to the *organisation.*

See Figure 10:

| Function | Entity | Source | Destntn | Move-one |
|---|---|---|---|---|
| *Record* | *Staff* | *Outsde W.* | *Organstn* | *move-many* |

Figure 10

## *Step 6 - Categorise each Entity*

So far little has been said about the nature of entities identified in Step 3. Step 6 suggests some features of these entities by categorising them. Select categories which describe the role of entities in the Production Planning domain. Three categories are:

* resource: the entity acts as a resource with which system requirements are fulfilled. Resources are often contained in a resource-container,
* resource-container: the entity is a container in which other entities are held,
* different-object-types: each instance of the entity may have many different values which play an important role in processing the entity, for example in a cinema seating domain both the reservation and the seat must be the same type (less than £3, etc).

You should assign one category to each entity and add these categories to Step 6 on the Answer Sheet. Note that not all entities will be described by one of these three categories.

*Example*
None of the above problem categories aptly describe the Staff, Organisation and Outside-world entities, so no entity categories are selected.

## *Step 7 - Identify Conditions on System Functions*

Functions sometimes often only occur when certain conditions are met. Consider each function identified in Step 2 and, where appropriate, select one of the following conditions which best describes the conditions under which the function occurs:

* Minimum-qty: the function occurs when an entity has reached a minimal level of contents,
* Maximum-qty: the function occurs when an entity has reached a maximum level of contents,
* Same-properties: the function occurs when two entities have the same values, for example a theatregoer is allocated to a seat if it meets his needs, i.e. seat price and his requirements are both < £20,
* Date/Time-limit: the function only occurs when a given date or time is reached, or after a specific length of time has passed.

Each function may only have one condition value, and a maximum of two functions may have conditions linked to them. Add each function and its condition to the list in Step 7 on the Answer Sheet.

*Example*
The Personnel system only records the coming and going of employees, so the information system has no specific conditions in the above list which control system functions.

## *Summary*

You should now have developed an appropriate model of the Production Planning domain. Reexamine this model to correct any omissions or contradictions. Specifically:

* Check the domain structure modelled in Step 4,
* Check the functional descriptions given in Step 5.

Finally inform the experimenter that you have completed your model of the Production Planning domain.

| *Name:* | **Answer Sheet** |
|---|---|

*Step 1- Major System Goal:*

*Step 2- Select functions by underlining them:*

| | | | | |
|---|---|---|---|---|
| Loan | Return | Input | Assign | Record |
| Borrow | Lend | Goods-in | Place | Finish-loan |
| Dispatch | Goods-out | Addition | Connect | Change-allocation |
| Send | Receipt | Allocate | Join | Change-position |

*Step 3- Identify Domain Entities:*

| |
|---|
| *Entity:* |
| *Entity:* |
| *Entity:* |
| *Entity:* |

*Step 4-Domain Structure:*

| *Entity-1* | *Entity-2* | *Relation* |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| WORLD | | |
| WORLD | | |
| WORLD | | |
| WORLD | | |

*Sketch:*

*Step 5-Function Definitions:*

| *Function* | *Entity* | *Source* | *Desitntn* | *move-one/ move-many* |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

*Sketch:*

*Step 6- Identify Entity Types:*

| *Entity* | *Type* |
|---|---|
| | |
| | |

| *Entity* | *Type* |
|---|---|
| | |
| | |

*Step 7- Identify Conditions on System Functions*

| *Function* | *Condition* |
|---|---|
| | |
| | |

**Question 1**

*Please identify how easy you found it to describe the production planning domain with each of the problem descriptors provided for you (circle a number). Also make comments on these descriptors where appropriate:*

*(a) The Object-relation Structure, e.g. Organisation contains-many Staff*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(b) Entity Categories e.g. Entity-A is a resource*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(c) Functional Descriptions e.g. move_many Staff from Outside-world to Organisation*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(d) Functions, e.g. Record*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(e) Functions beyond the system scope, e.g. Record*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(f) System requirements*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(g) Labels, e.g. stock control system*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

## Question 2

(a) *Did you find it easy to use the windows and dialogues provided by the tool to describe the production planning domain (if not, why not) ?*

(b) *Did you enter data in the order suggested by the tool, or did you use the pull-down menus to add facts about the domain after the appropriate window had been quit ?*

## Question 3

*How easy or difficult did you find it to use and understand the example Personnel domain provided (circle a number). Comment on any difficulties encountered while using the example:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| *very difficult* | | | | | | *very easy* |

*Comment:*

## Question 4

*Were you able to describe all the features of the Production Planning that you wanted to describe ? If not, what other features would you like to have mentioned ?*

## Question 5

*What other help would you like to have received when trying to describe the production planning domain with the instructions provided ?*

**Question 6**

*Please identify how easy you found it to use the following features of the Ira toolkit (circle a number). Also make comments on these descriptors where appropriate:*

*(a) the windows describing the different knowledge types elicited by Ira (e.g. the 'Introduction to Ira' window):*

1 2 3 4 5 6 7

very difficult — very easy

Comment:

*(b) the dialogues provided for inputting specific types of data (e.g. the dialogue eliciting object-relations):*

1 2 3 4 5 6 7

very difficult — very easy

Comment:

*(c) the 'See Target' window which described the current description of the new problem:*

1 2 3 4 5 6 7

very difficult — very easy

Comment:

*(d) the 'General Help' 'window describing an overview of the windows provided by the tool:*

1 2 3 4 5 6 7

very difficult — very easy

Comment:

*(e) the abstraction domain descriptions retrieved by Ira's search mechanism:*

1 2 3 4 5 6 7

very difficult — very easy

Comment:

*(f) the analogical mappings identified by Ira's search mechanism:*

1 2 3 4 5 6 7

very difficult — very easy

Comment:

**Question 8**

*Please give your reasons for the following facts identified about the production planning domain:*

| *Entity* | *Relation* | *Entity* | *Reason* |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

| *Entity* | *Entity-Category* | *Reason* |
|---|---|---|
| | | |
| | | |

| | *Entity* | *Source* | *Destination* | *(Condition)* | *Reason* |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

| *Function* | *Reason* |
|---|---|
| | |
| | |

| *Other Features* | *Reason* |
|---|---|
| | |
| | |
| | |
| | |
| | |

# Appendix F - Experimental Material for Empirical Investigation of the Prototype Problem Identifier Module

**Problem: Production Planning in the Roker Manufacturing Company**

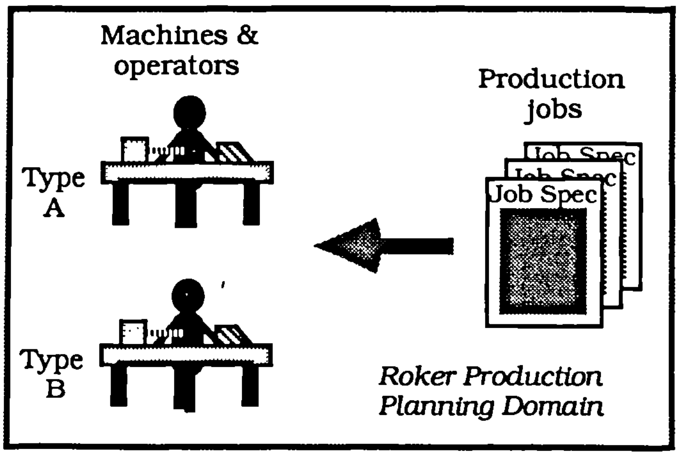
Roker Manufacturing is a medium-sized company making heavy equipment for shipbuilding. Their plant consists of several workshops, each containing specialised machines for the construction of different types of equipment. Production is planned monthly by allocating manufacturing jobs to appropriate machines in the workshops. This document focuses on the monthly production planning process. You are required to analyse this process and model it using techniques described in the accompanying document. Do not consider other aspects of the Roker manufacturing system.

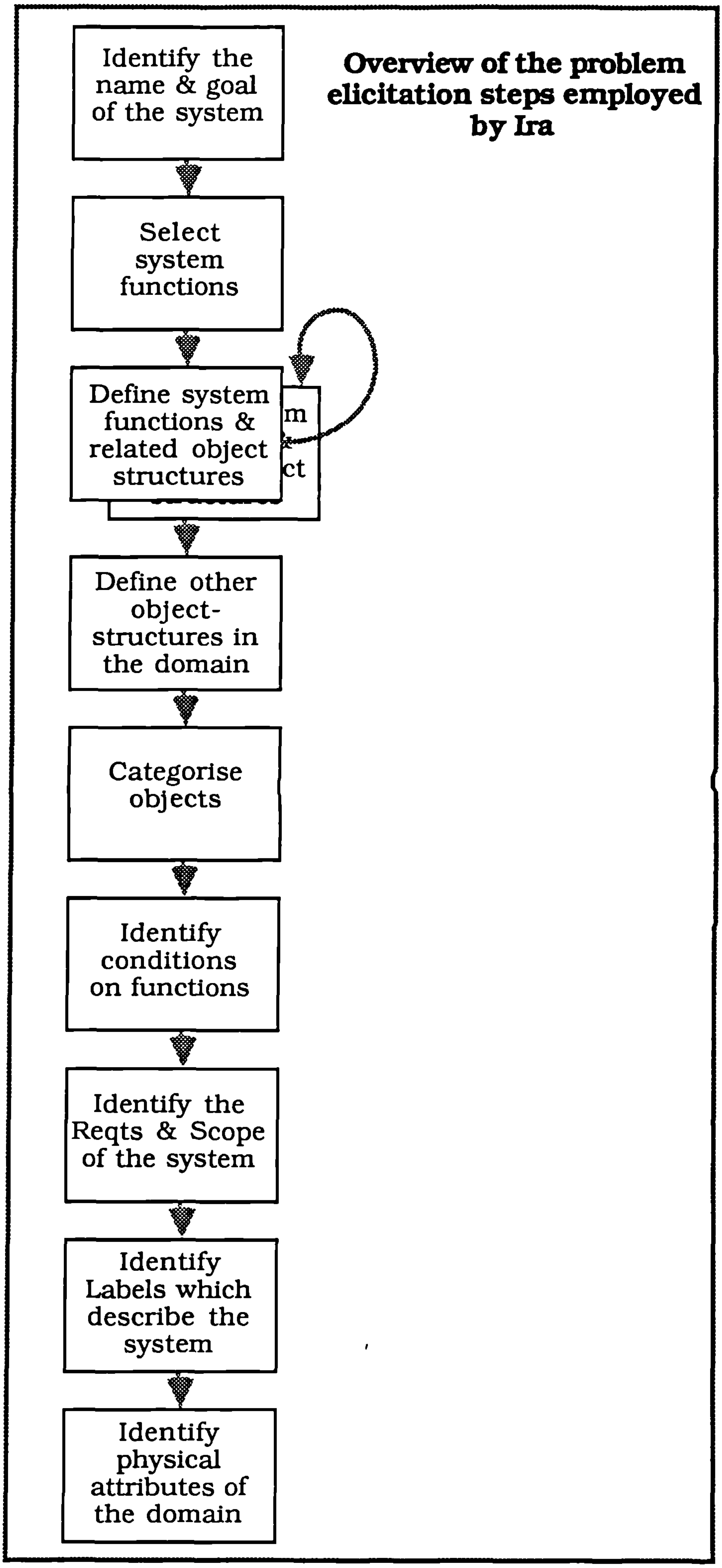
The production planning process is carried out at the beginning of every month by a computerised scheduling system. The aim of this system is to allocate the monthly quota of manufacturing jobs to machines in a way that:
* maximises the use of machines,
* ensures that a maximum number of jobs are completed by their deadline.

The allocation of a job to a machine is constrained by several important limitations, including:
* the suitability of a machine to manufacture a product,
* the availability of skilled operators to use machines,
* the sequence in which manufacturing jobs must be carried out,
* the ability of certain machines to complete a job in the time allowed.

In addition, certain jobs are given priority allocation to machines. Priority jobs occur when:
* the necessary machines are rare,
* the necessary skilled operators are rare.

In this study you are required to describe facts about the production planning domain by following instructions given in the Ira Toolkit. To enter Ira use the instructions given on the accompanying sheet.




Overview of the problem elicitation steps employed by Ira
Identify the name & goal of the system
Select system functions
Define system functions & related object structures
Define other object-structures in the domain
Categorise objects
Identify conditions on functions
Identify the Reqts & Scope of the system
Identify Labels which describe the system
Identify physical attributes of the domain

**Retrospective Questioning: Question 1**

*Please identify how easy you found it to describe the production planning domain with each of the problem descriptors provided for you (circle a number). Also make comments on these descriptors where appropriate:*

*(a) The Object-relation Structure, e.g. Organisation contains-many Staff*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(b) Entity Categories e.g. Entity-A is a resource*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(c) Functional Descriptions e.g. move_many Staff from Outside-world to Organisation*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(d) Functions, e.g. Record*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(e) Functions beyond the system scope, e.g. Record*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(f) System requirements*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

*(g) Labels, e.g. stock control system*

| 1 very difficult | 2 | 3 | 4 | 5 | 6 | 7 very easy |
|---|---|---|---|---|---|---|

*Comment:*

**Question 2**

(a) *Did you find it easy to use the windows and dialogues provided by the tool to describe the production planning domain (if not, why not) ?*

(b) *Did you enter data in the order suggested by the tool, or did you use the pull-down menus to add facts about the domain after the appropriate window had been quit ?*

**Question 3**

*How easy or difficult did you find it to use and understand the example Personnel domain provided (circle a number). Comment on any difficulties encountered while using the example:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| *very difficult* | | | | | | *very easy* |

*Comment:*

**Question 4**

*Were you able to describe all the features of the Production Planning that you wanted to describe ? If not, what other features would you like to have mentioned ?*

**Question 5**

*What other help would you like to have received when trying to describe the production planning domain with the instructions provided ?*

**Question 6**

*Please identify how easy you found it to use the following features of the Ira toolkit (circle a number). Also make comments on these descriptors where appropriate:*

*(a) the windows describing the different knowledge types elicited by Ira (e.g. the 'Introduction to Ira' window):*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| very difficult | | | | | | very easy |

Comment:

*(b) the dialogues provided for inputting specific types of data (e.g. the dialogue eliciting object-relations):*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| very difficult | | | | | | very easy |

Comment:

*(c) the 'See Target' window which described the current description of the new problem:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| very difficult | | | | | | very easy |

Comment:

*(d) the 'General Help' 'window describing an overview of the windows provided by the tool:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| very difficult | | | | | | very easy |

Comment:

*(e) the abstraction domain descriptions retrieved by Ira's search mechanism:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| very difficult | | | | | | very easy |

Comment:

*(f) the analogical mappings identified by Ira's search mechanism:*

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| very difficult | | | | | | very easy |

Comment:

## Question 7

*(a) Did you modify your description of the problem domain as a result of feedback from the analogical search ? (If not, why ?)*

*(b) Were you able to select between the retrieved abstract domains which Ira retrieved from searching ? (If not, how ?)*

*(c) Did you have any other difficulties in understanding the abstract domain models retrieved by Ira ?*

**Question 8**

*Please give your reasons for the following facts identified about the production planning domain:*

| Entity | Relation | Entity | Reason |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

| Entity | Entity-Category | Reason |
|---|---|---|
| | | |
| | | |

| | Entity | Source | Destination | (Condition) | Reason |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

| Function | Reason |
|---|---|
| | |
| | |

| Other Features | Reason |
|---|---|
| | |
| | |
| | |
| | |
| | |

# Appendix G - Results from the Experimental Evaluation of the Analogy Engine using Instances of Partial Target Domain Descriptions

| Domain Matched | Abstract Domain Class | Test Without Structure | Test Without Transition |
|---|---|---|---|
| Stock Control System | OCP-BA | Perfect | Perfect |
| Personnel System | OCP-BB | Fail | Partial* |
| Library System | OCP-AB | Perfect | OCP Only |
| Air Traffic Control System | OMP | Fail | Perfect |
| Coastguard Patrol System | OPP | Fail | Fail |
| Simple Theatre System | OAP | Perfect | Perfect |
| Complex Theatre System | OAP-AA | Perfect | Perfect |

**Partial match with OCP, OMP & OAP classes*

Appendix G -
Results of First-pass Evaluation of the Example Search Space

# Appendix H - Bug Library of Errors made during Analogical Comprehension and Transfer of Specifications by Inexperienced Software Engineers

| Bug Description | | Evidence of Occurence |
|---|---|---|
| 1. | *Fail to return of omitted analogical mappings for completion* | *Transfer reusable processes with which analogical mappings were identified, ignore those which were not transferred* |
| 2. | *Fail to distinguish between similar reusable components in the solution specification due to syntactic similarities* | *Failure to transfer all reusable components which share syntactic similarities with other, recently transferred components* |
| 3. | *Abandon hypothesis about source domain concepts due to lack of source domain knowledge* | *Failure to transfer reusable components which share no other syntactic similarities with other reusable solution components* |
| 4. | *Develop required target components incorrectly, based on structural similarity of the reusable specification rather than underlying analogical understanding* | *Incorrect analogical transfer without any syntactic similarity implied by the erroneous analogical match* |
| 5. | *Failure to distinguish between source concepts which have similar roles in both domains* | *Failure to reuse all solution components due to equivalent roles in both domains* |
| 6. | *Incorrect solution component reuse due to direct syntactic similarity between target source components.* | *Incorrect reuse of components, with syntactic similarity between target and reused components* |
| 7. | *Incorrect solution component reuse due to indirect syntactic similarity between target source components.* | *Incorrect reuse of components, with syntactic similarity between target and reused components* |
| 8. | *Poor analogical reasoning* | *No external effect* |
| 9. | *Confuse synomynous source domain concepts* | *Incorrect component reuse with reusable components whose underlying source concepts are close to the correct analogical match* |

Bug library of analogical reuse errors exhibited by inexperienced software engineers during understanding and transfer of unfamiliar specifications

# Appendix I- Key Algorithms in the Analogy Engine

*Object Structural Knowledge ($A_o$ is Abstract object structure, $T_o$ is Target object structure):*

object_structure_mapping ($A_o$ , $T_{oo}$ ) <--
alternative_target_structure ( $T_o$ , $T_{oo}$ ),
same_relation_type ( $A_o$ , $T_{oo}$ ),
correct_level_objects ( $T_{oo}$ ),
best_candidate_mapping ( $T_{oo}$ , $T_{others}$ ).

*State transitions ($A_t$ is Abstract state transition, $T_t$ is Target state transition):*

state_transition_mapping ( $A_t$ , $T_t$ ) <--
correct_level_objects ( $T_t$ ),
same_transition_type ( $A_t$ , $T_t$ ),
best_candidate_mapping ( $T_t$ , $T_{others}$ ).

*Best candidate mapping is defined as for mappings between object structure knowledge and state transitions is:*

best_candidate_mapping ( T , Others ) <--

count_neighbouring_matches ( T , $Count_t$ ),
count_neighbouring_matches ( T , $Count_{others}$ ),
$Count_t$ > $Count_{others}$.

*Object type Mappings:*

object_type_mapping ( $A_o$ , $T_o$ ) <--
correct_level_objects ( $T_o$ ),
object_structure_mapping ( $A_o$ , $T_o$ ),
same_object_type ( $A_o$ , $T_o$ ),!.

object_type_mapping ( $A_t$ , $T_t$ ) <--
correct_level_objects ( $T_t$ ),
state_transition_mapping ( $A_t$ , $T_t$ ),
same_object_type ( $A_t$ , $T_t$ ).

where additional constraints on the matching heuristics are:

correct_level_objects ( $T_o$ ) <--!

alternative_target_structure ( To, $T_{oo}$ ) <--!

## Revised structural coherence algorithms to incorporate matching within a hierarchy of abstract domain classes

*Object Structural Knowledge ($A_o$ is Abstract object structure, $T_o$ is Target object structure):*

object_structure_mapping ($A_o$ , $T_o$ ) <--
same_relation_type ( $A_o$ , $T_o$ ),
best_candidate_mapping ( $T_o$ , $T_{others}$ ).

*State transitions ($A_t$ is Abstract state transition, $T_t$ is Target state transition):*

state_transition_mapping ( $A_t$ , $T_t$ ) <--
same_transition_type ( $A_t$ , $T_t$ ),
best_candidate_mapping ( $T_t$ , $T_{others}$ ).

*Best candidate mapping is defined as for mappings between object structure knowledge and state transitions is:*

best_candidate_mapping ( T , Others ) <--

count_neighbouring_matches ( T , $Count_t$ ),
count_neighbouring_matches ( T , $Count_{others}$ ),
$Count_t > Count_{others}$.

*Object type Mappings:*

object_type_mapping ( $A_o$ , $T_o$ ) <--
object_structure_mapping ( $A_o$ , $T_o$ ),
same_object_type ( $A_o$ , $T_o$ ),!.

object_type_mapping ( $A_t$ , $T_t$ ) <--
state_transition_mapping ( $A_t$ , $T_t$ ),
same_object_type ( $A_t$ , $T_t$ ).

## Algorithms to determine structural coherence between two isolated domain models

Perfect_Coherence_Structure ( $A_m$ , $T_m$ ) <--
Addup_Mappings ( $T_m$ , $Mappings_t$ ),
Total_Mappings ( $A_m$ , $Mappings_m$ ),
( $Mappings_t$ / $Mappings_m$ ) ≥ 81%.

Good_Coherence_Structure ( $A_m$ , $T_m$ ) <--
Addup_Mappings ( $T_m$ , $Mappings_t$ ),
Total_Mappings ( $A_m$ , $Mappings_m$ ),
50% ≥ ( $Mappings_t$ / $Mappings_m$ ) < 81%.

Effective_Abstract_Difference ( $A_m$ , $T_m$ ) <--
Addup_Differences ( $T_m$ , $Differences_t$ ),
Total_Differences ( $A_m$ , $Differences_m$ ),
( $Differences_t$ / $Differences_m$ ) > 33%.

Effective_Alternative_Match ( $A_m$ , $T_m$ ) <--'
System_Requirement_Mapping ( ($A_o$,$A_r$) , ($T_o$,$T_r$) ),
System_Scope_Mapping ( ($A_t$,$A_s$) , ($T_t$,$T_s$) ),!.

Effective_Alternative_Match ( $A_m$ , $T_m$ ) <--
Addup_Terms ( $T_m$ , $Terms_t$ ),

Total_Terms ( $A_m$ , $Terms_m$ ),
( $Terms_t$ /$Terms_m$ ) > 66%.

Figure - algorithms determining the degree of structural coherence and critical difference between candidate abstract domain classes

System_Requirement_Mapping ( ($A_o$,$A_r$) , ($T_o$,$T_r$) ) <--
object_structure_mapping ( $A_o$ , $T_o$ ),
same_requirement_type ( $A_r$ , $T_r$ ).

System_Scope_Mapping ( ($A_t$,$A_s$) , ($T_t$,$T_s$) ) <--
state_transition_mapping ( $A_t$ , $T_t$ ),
same_scope_type ( $A_s$ , $T_s$ ).

Figure - algorithms determining analogical similarity between system requirements and information system scope

Good_Match ( $A_m$ , $T_m$ ) <--
Perfect_Coherence_Structure ( $A_m$ , $T_m$ ),!.

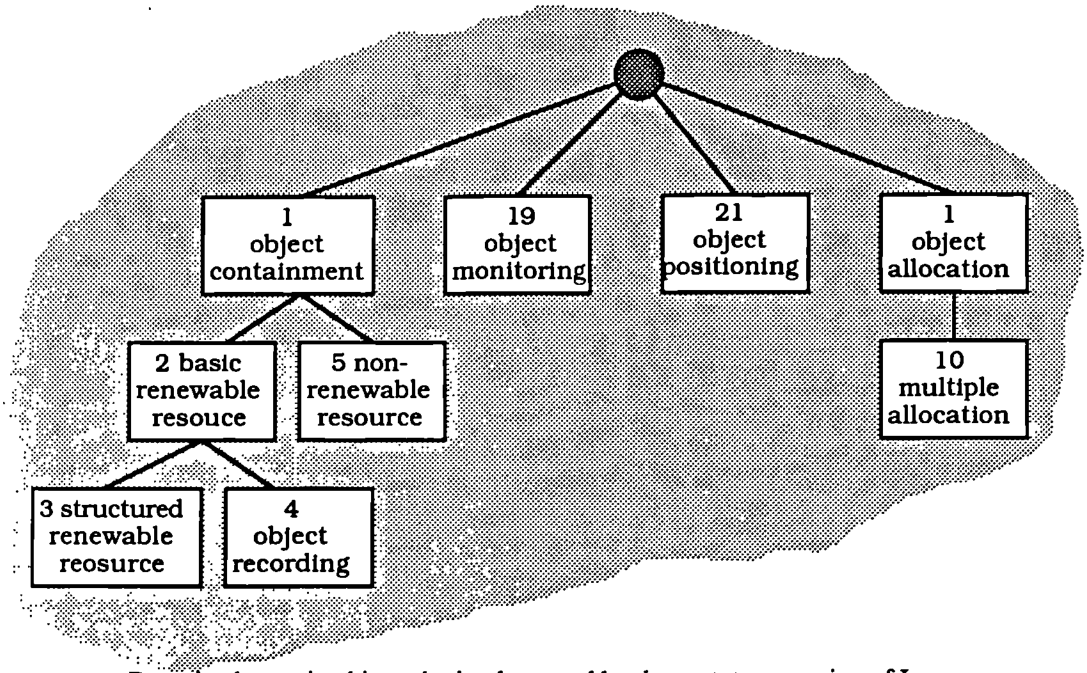
Good_Match ( $A_m$ , $T_m$ ) <--
Good_Coherence_Structure ( $A_m$ , $T_m$ ),
Effective_Abstract_Difference ( $A_m$ , $T_m$ ),!.

Good_Match ( $A_m$ , $T_m$ ) <--
Good_Coherence_Structure ( $A_m$ , $T_m$ ),
Effective_Alternative_Match ( $A_m$ , $T_m$ ).

Partial_Match ( $A_m$ , $T_m$ ) <--
Good_Coherence_Structure ( $A_m$ , $T_m$ ).

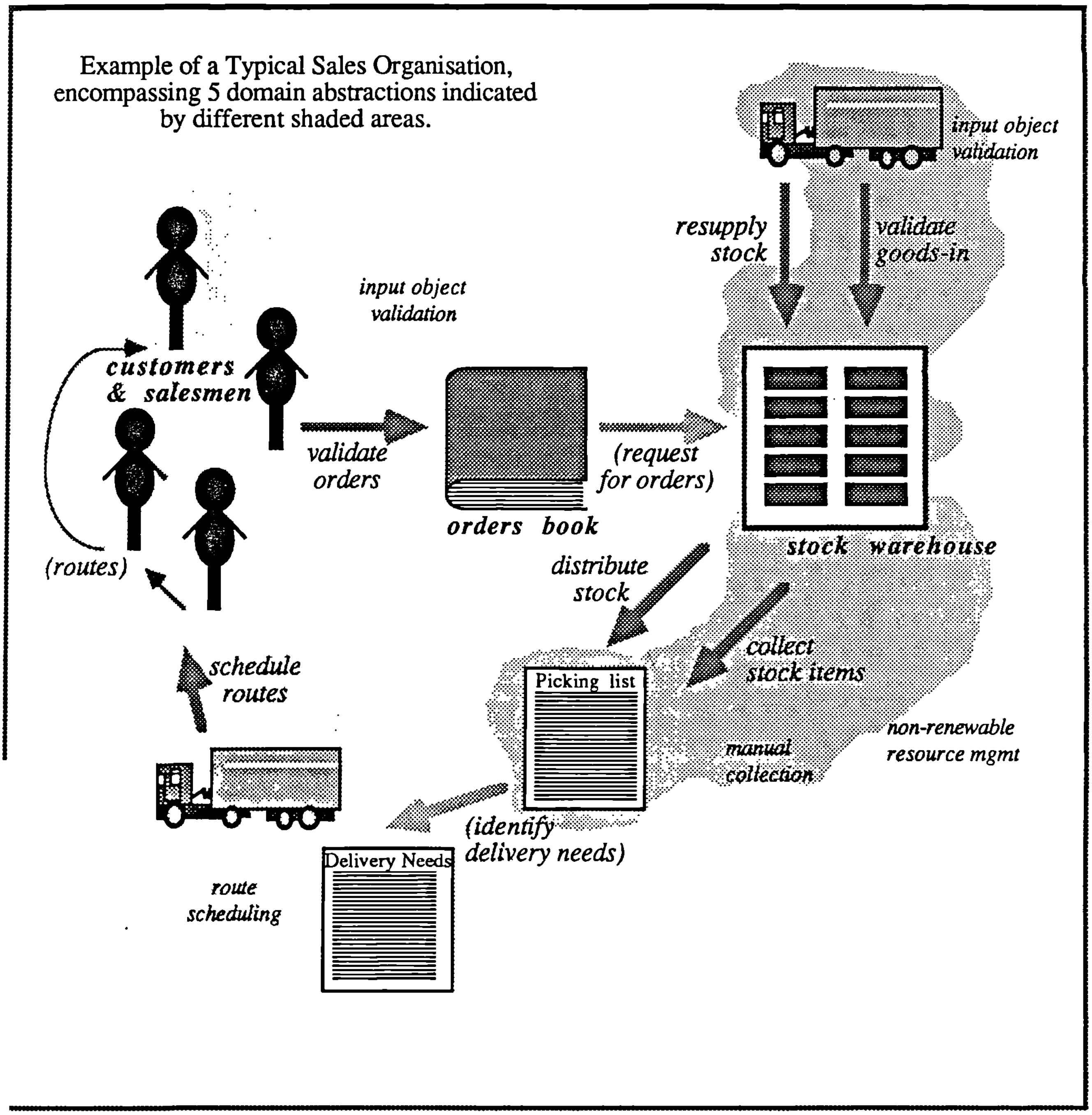
Figure - algorithms determining a good or partial analogical fit from the degree of structural fit and the extent of critical differences between abstractions

# Appendix J - Domain Abstraction Hierarchy Implemented by the Analogy Engine

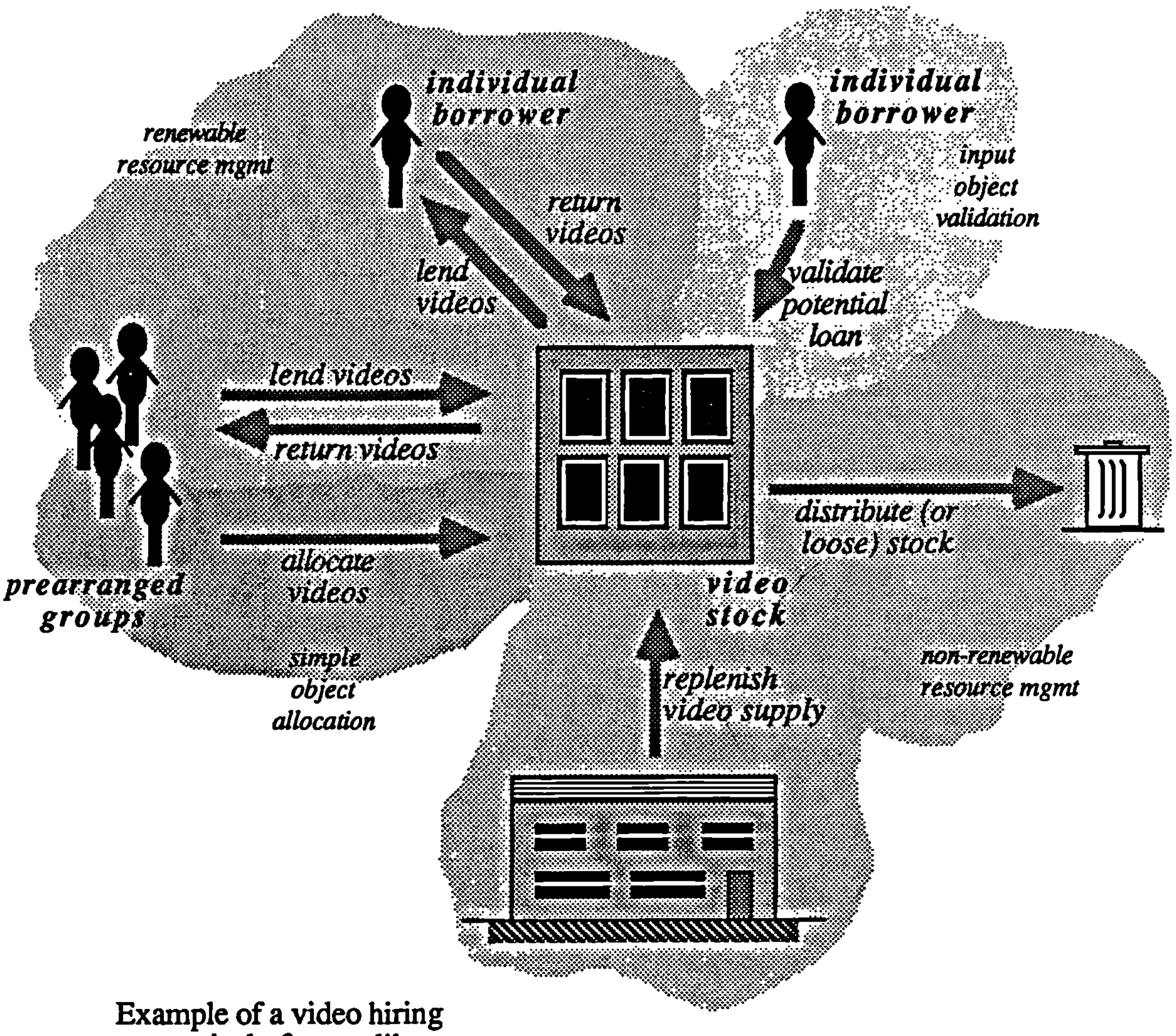



Domain abstraction hierarchy implemented by the prototype version of Ira

# Appendix K - Aggregated Domain Abstractions Representing Common, Larger Domain Types




Example of a Typical Sales Organisation, encompassing 5 domain abstractions indicated by different shaded areas.




Example of a video hiring store, typical of many library-type organisations involved with maintenance of a lending stock

# Appendix L - Source Listing of Prototype Implementation of Ira, using Prolog

# Descriptions of the Domain Abstractions

```
/* Description of all aspects of the ocp, omp and oap ACPs */

/* Static Knowledge Descriptors */

acp_sdata(space,slot,has_one,ocp).
acp_sdata(space,space2,has_many,ocp).
acp_sdata(slot,object,contains_many,ocp).
acp_sdata(space2,object,has_many,ocp).

acp_sdata(space,slot,has_many,omp).
acp_sdata(slot,object,contains_no,omp).
acp_sdata(slot,object,contains_one,omp).
acp_sdata(slot,object,contains_many,omp).
acp_sdata(space,object,has_no,omp).

acp_sdata(space,allocation,has_one,oap).
acp_sdata(space,object,has_many,oap).
acp_sdata(allocation,object,contains_many,oap).

acp_sdata(allocation,slot,has_many,oapaa).
acp_sdata(slot,object,contains_no,oapaa).
acp_sdata(slot,object,contains_one,oapaa).

acp_sdata(space,space1,has_one,ocpaa).
acp_sdata(space1,object,has_many,ocpaa).

acp_sdata(smallslot,object,contains_many,ocpba).
acp_sdata(slot,smallslot,has_many,ocpba).

acp_sdata(space,slot,has_one,opp).
acp_sdata(space1,object,has_one,opp).
acp_sdata(space2,object,has_one,opp).
acp_sdata(slot,object,contains_no,opp).
acp_sdata(slot,object,contains_one,opp).

/* Dynamic knowledge Descriptors */

acp_ddata(dispatch,object,slot,space2,move_many,ocp).
acp_ddata(receipt,object,space1,slot,move_many,ocpaa).
acp_ddata(return,object,space2,slot,move_many,ocpab).
acp_ddata(receipt,object,space1,smallslot,move_many,ocpba).
acp_ddata(dispatch,object,smallslot,space2,move_many,ocpba).
acp_ddata(monitor,object,slot,slot,move_one,omp).
acp_ddata(allocate,object,space,allocation,move_many,oap).
acp_ddata(allocate,object,space,slot,move_many,oapaa).
acp_ddata(monitor,object,space1,slot,move_one,opp).
acp_ddata(monitor,object,slot,space2,move_one,opp).

/* Property features of each object/slot in the system. */

acp_pdata(ocp,object,resource).
acp_pdata(ocp,slot,resource_container).
acp_pdata(ocpba,smallslot,resource_container).
acp_pdata(oap,object,different_object_types).
acp_pdata(oap,allocation,recepticable).
acp_pdata(oapaa,slot,different_object_types).

/* Condition Knowledge Descriptors */
```

```
acp_cdata(return,date_limit,ocpab).
acp_cdata(receipt,minimum_qty,ocpba).
acp_cdata(allocate,same_properties,oapaa).

/* Scope of influence of the computerised system for each ACP -
   all scope refers to the movement of objects, no other
   knowledge types - it identifies what is beyond the scope of
   the computerised system */

acp_scope(dispatch,ocp).
acp_scope(return,ocpab).
acp_scope(monitor,opp).
acp_scope(monitor,opp).
acp_scope(monitor,omp).
acp_scope(record,ocpbb).

/* Requirements relate to the static knowledge
   structures, and include additional features to identify needs */

acp_reqt(slot,object,contains_many,minimum_qty,ocp).
acp_reqt(slot,object,contains_many,minimum_qty,ocpaa).
acp_reqt(space2,object,contains_many,date_limit,ocpab).
acp_reqt(smallslot,object,contains_many,minimum_qty,ocpba).
acp_reqt(slot,object,contains_one,opp).
acp_reqt(slot,object,contains_no,omp).
acp_reqt(slot,object,contains_one,omp).
acp_reqt(object,space,has_no,oap).
acp_reqt(slot,object,contains_one,same_properties,oapaa).

acp_reqt_total(ocp,4).
acp_reqt_total(ocpaa,1).
acp_reqt_total(ocpab,1).
acp_reqt_total(ocpba,1).
acp_reqt_total(ocpbb,0).
acp_reqt_total(omp,2).
acp_reqt_total(oap,1).
acp_reqt_total(oapaa,1).

/* Objects describing each ACP - 'Space' objects are not included. */

acp_object(object,ocp).
acp_object(slot,ocp).
acp_object(object,ocpaa).
acp_object(slot,ocpaa).
acp_object(object,ocpab).
acp_object(slot,ocpab).
acp_object(object,ocpba).
acp_object(slot,ocpba).
acp_object(smallslot,ocpba).
acp_object(object,ocpbb).
acp_object(slot,ocpbb).
acp_object(object,opp).
acp_object(slot,opp).
acp_object(object,omp).
acp_object(slot,omp).
acp_object(object,oap).
acp_object(allocation,oap).
acp_object(slot,oap).
acp_object(object,oapaa).
```

```
acp_object(allocation,oapaa).
acp_object(slot,oapaa).

/* Physical features of specific acps in the problem space */

acp_phyprop(object,are_borrowed,ocpab).
acp_phyprop(object,taken_away,ocpab).
acp_phyprop(slot,is_building,ocpba).
acp_phyprop(smallslot,in_building,ocpba).
acp_phyprop(smallslot,is_container,ocpba).
acp_phyprop(smallslot,different_properties,ocpba).
acp_phyprop(object,different_properties,ocpba).
acp_phyprop(object,moves_physically,opp).
acp_phyprop(object,are_manned_vehicle,opp).
acp_phyprop(slot,adjacent_in_space,opp).
acp_phyprop(object,moves_physically,omp).
acp_phyprop(slot,in_sequence,omp).
acp_phyprop(slot,construct_network,omp).

/* Total of physical structures in each ACP. */

acp_phymappings(ocp,0).
acp_phymappings(ocpaa,0).
acp_phymappings(ocpab,2).
acp_phymappings(ocpba,5).
acp_phymappings(ocpbb,0).
acp_phymappings(opp,3).
acp_phymappings(omp,3).
acp_phymappings(oap,0).
acp_phymappings(oapaa,0).

/* Other knowledge supporting the identification of critical differences
   between the acps */

acp_data(information_system,ocp).
acp_data(safety_critical,omp).

/* Labels allocated to each acp to describe different facets of each acp -
   there are three labels allocated to each ACP */

acp_label(stock_control,ocp).
acp_label(object_containment,ocp).
acp_label(resource_management,ocp).
acp_label(stock_control,ocpaa).
acp_label(object_containment,ocpaa).
acp_label(renewable_resource_management,ocpaa).
acp_label(library_system,ocpab).
acp_label(non-renewable_resource_management,ocpab).
acp_label(object_hiring,ocpab).
acp_label(object_recording,ocpbb).
acp_label(organisation_content,ocpbb).
acp_label(personnel,ocpbb).
acp_label(stock_control,ocpba).
acp_label(object_containment,ocpba).
acp_label(resource_management,ocpba).
acp_label(space_occupation,opp).
acp_label(single_object_containment,opp).
acp_label(space_management,opp).
acp_label(object_monitoring,omp).
```

```
acp_label(collision_detection,omp).
acp_label(plan_adherence,omp).
acp_label(object_allocation,oap).
acp_label(constraint_satisfaction,oap).
acp_label(requirement_matching,oap).
acp_label(object_allocation,oapaa).
acp_label(constraint_satisfaction,oapaa).
acp_label(requirement_matching,oapaa).

/* Hierarchical Structure of the ACPs */

father(ocp,ocpaa).
father(ocp,ocpab).
father(ocpaa,ocpba).
father(ocpaa,ocpbb).
father(top,ocp).
father(top,omp).
father(top,oap).
father(oap,oapaa).
father(top,opp).

/* Total number of possible analogical mappings with each ACP, and the
   total number of possible differences. */

acp_total_mappings(ocp,9).
acp_total_mappings(ocpaa,5).
acp_total_mappings(ocpab,4).
acp_total_mappings(ocpba,8).
acp_total_mappings(ocpbb,0).
acp_total_mappings(opp,11).
acp_total_mappings(omp,5).
acp_total_mappings(oap,8).
acp_total_mappings(oapaa,7).

/* Note - when calculating these totals the number of possible function
   mappings for each ACP is considered so that the full possible number
   of ACP differences is known - functional transformations introduce a
   degree of variability into these totals. */

acp_total_differences(ocp,11).
acp_total_differences(ocpaa,5).
acp_total_differences(ocpab,5).
acp_total_differences(ocpba,1).
acp_total_differences(ocpbb,3).
acp_total_differences(opp,8).
acp_total_differences(omp,10).
acp_total_differences(oap,8).
acp_total_differences(oapaa,0).

/* Valid Acps and names, including a dummy to assist in dialogue list
   construction. */

acps('','').
acps(ocp,'Object Containment Problem').
acps(ocpaa,'Non-renewable Resource Mgmt Problem').
acps(ocpab,'Renewable Resource Mgmt Problem').
acps(ocpba,'Structured Non-renewable Resource Mgmt Problem').
acps(ocpbb,'Object Recording Problem').
acps(opp,'Single Object Spatial Management Problem').
```

```
acps(omp,'Object Monitoring Problem').
acps(oap,'Object Allocation Problem').
acps(oapaa,'Constrained Object Allocation Problem').

/* The high-level functional thesaurus, which retrieves correct and other
   most likely example model for each function which was input. One
   input exists for each function in the thesaurus. */

mainfunctions(loan,ocp,oap).
mainfunctions(borrow,ocp,oap).
mainfunctions(dispatch,ocp,oap).
mainfunctions(send,ocp,opp).
mainfunctions(lend,ocp,oap).
mainfunctions(goods_out,ocp,oap).
mainfunctions(receipt,ocp,oap).
mainfunctions(input,ocp,oap).
mainfunctions(goods_in,ocp,opp).
mainfunctions(arrival,ocp,omp).
mainfunctions(addition,oap,ocp).
mainfunctions(allocate,oap,ocp).
mainfunctions(assign,oap,opp).
mainfunctions(place,oap,ocp).
mainfunctions(correct,oap,ocp).
mainfunctions(join,oap,ocp).
mainfunctions(return,ocp,oap).
mainfunctions(finish_loan,ocp,oap).
mainfunctions(check_position,omp,opp).
mainfunctions(monitor,omp,opp).
mainfunctions(record,ocp,oap).
```

```
/* This program resets the recorded analogous mappings and the target
   domain data base through commands accessed on the 'Run Ira menu'. */

/* Series of heuristics to remove all generated analogous mappings. A
   dummy rule is included that the rule fires even when no mappings
   exist in the data base. Findall is necessary to ensure that all deletions
   are carried out in the one call to a routine. */

removeall_mappings :-
disable_item('Control','Identify Mappings'),
banner(findall(_,remove_eachmapping,_),['Please be patient - Ira is removing all previous analogous
mappings'],150,110).

remove_eachmapping :-
retractall(rec_acpmatch(_)).
remove_eachmapping :-
retractall(rec_statmapping(_,_,_,_,_,_)).
remove_eachmapping :-
retractall(rec_dynmapping(_,_,_,_,_,_,_,_)).
remove_eachmapping :-
retractall(rec_propmapping(_,_,_,_)).
remove_eachmapping :-
retractall(rec_condmapping(_,_,_,_)).
remove_eachmapping :-
retractall(rec_funcmapping(_,_)).
remove_eachmapping :-
retractall(rec_objectmatch(_,_,_,_)).
remove_eachmapping :-
retractall(rec_reqtmapping1(_,_,_,_,_,_)).
remove_eachmapping :-
retractall(rec_reqtmapping2(_,_,_,_,_,_,_)).
remove_eachmapping :-
retractall(rec_scopemapping(_,_,_,_,_,_,_,_)).
remove_eachmapping :-
retractall(rec_funcmapping(_,_)).
remove_eachmapping :-
retractall(rec_labelmapping(_,_)).
remove_eachmapping :-
retractall(rec_phymapping(_,_,_,_)).
remove_eachmapping :- !.

/* Series of heuristics to remove all target data from data base. A dummy
   rule is included to ensure firing when no target rules exist. */

removeall_target :-
retractall(target_object(_)).
removeall_target :-
retractall(target_sdata(_,_,_)).
removeall_target :-
retractall(target_ddata(_,_,_,_,_)).
removeall_target :-
retractall(target_cdata(_,_)).
removeall_target :-
retractall(target_pdata(_,_)).
removeall_target :-
retractall(target_scope(_,_)).
removeall_target :-
retractall(target_reqt(_,_,_)).
removeall_target :-
```

```
retractall(target_reqt(_,_,_,_)).
removeall_target :-
retractall(target_phyprop(_,_)).
removeall_target :-
retractall(target_label(_)).
removeall_target :-
retractall(target_goal(_)).
removeall_target :-
retractall(target_name(_)).
removeall_target :-
assertz(target_object(world)).
```

# Descriptions of the Matching Algorithms of the Analogy Engine

```
/* Program to examine the critical differences between  ACPs, looking
   at each selected_acp in turn - the critical differences are implicitly
   built into the rules */

/* First layer of rules identifies the level of success for each Acp. */

perfect_difference(Selected_acp) :-
calc_difference(Selected_acp,Diffscore),
Diffscore >= 80.

good_difference(Selected_acp) :-
calc_difference(Selected_acp,Diffscore),
Diffscore >= 50,
Diffscore < 80.

poor_difference(Selected_acp) :-
calc_difference(Selected_acp,Diffscore),
Diffscore >= 0,
Diffscore < 50.

fail_difference(Selected_acp) :-
calc_difference(Selected_acp,Diffscore),
Diffscore < 0.

/* First-level comparision program which identifies the scale of
   difference between problems, to decide whether the difference is
   effective for the matching process. It analyses all scores for all
   abstractions in the search space, sorts them, then checks the difference
   to establish a gap of at least 33% between the best and other
   abstractions. A special rule is included to permit the OAPAA to pass this
   rule successfully, since the generic version of the rule fails with only
   one son at the level of the hierarchy. */

effective_difference(oapaa) :- !.

effective_difference(Selected_acp) :-
findall((Score,Candidate_acp),(
father(Father,Selected_acp),
father(Father,Candidate_acp),
calc_difference(Candidate_acp,Score)),
Acplist),
sort(Acplist,Newlist,[],1),
Newlist=[(Score1,Selected_acp),(Score2,Second_acp)|Rest],
Difference is Score1-Score2,
Difference>=33.

/* Second layer of rules to count the totals of good and bad differences
   for each selected_acp. A specific version of the rule is given for OAPAA
   since it has 0 differences, and the program must not be allowed to
   divide by zero. */

calc_difference(oapaa,0) :- !.

calc_difference(Selected_acp,Diffscore) :-
good_diffscore(Selected_acp,Difftotal),
acp_total_differences(Selected_acp,Total),
Diffscore is (Difftotal/Total)*100.

/* Third layer of rules to develop a list of good matches and to count
```

```
   this list to give a total of all good difference matches. */

good_diffscore(Selected_acp,Good_score) :-
findall(Diffs,difference_matches(Diffs,Selected_acp),Diff_list),
length(Diff_list,Good_score).

/* Fourth layer of rules to identify the good differences for each acp
   in turn. Double-strength, important differences are possible if the
   difference rule is repeated and an additional DIFF is put into the
   Diff-list. */

/* Differences to distinguish OCP, OMP, OAP & OPP structures. Several
   complex rules are used here to assist in the matching process.  */

/* Difference 1- the number of slots in the world. */

difference_matches(Diffs,ocp) :-
static_mapping(_,_,space,slot,has_one,Score,ocp),
Diffs = m1.

difference_matches(Diffs,oap) :-
static_mapping(_,_,space,allocation,has_one,Score,oap),
Diffs = m1.

difference_matches(Diffs,opp) :-
static_mapping(_,_,space,slot,has_one,Score,opp),
Diffs = m1.

difference_matches(Diffs,omp) :-
static_mapping(_,_,space,slot,has_many,Score,omp),
Diffs = m1.

/* Difference 2- the number of objects held in the external world. */

difference_matches(Diffs,omp) :-
static_mapping(_,_,space,object,has_no,Score,omp),
Diffs = m2.

difference_matches(Diffs,opp) :-
static_mapping(_,_,space,object,has_one,Score,opp),
Diffs = m2.

difference_matches(Diffs,ocp) :-
static_mapping(_,_,space2,object,has_many,Score,ocp),
Diffs = m2.

difference_matches(Diffs,oap) :-
static_mapping(_,_,space,object,has_many,Score,oap),
Diffs = m2.

/* Difference 3- the scope of object movements external to the world. */

difference_matches(Diffs,omp) :-
target_scope(_,Obj1,Obj2,Obj3,move_one),
object_matching(Obj1,object,_),
object_matching(Obj2,slot,_),
object_matching(Obj3,slot,_),
Diffs = m3.
```

```
/* Difference 4- safety-critical systems. */
/*
difference_matches(Diffs,ocp) :-
not target_data(safety_critical),
Diffs = m4.
*/
/* Difference 5- real-time systems. */
/*
difference_matches(Diffs,omp) :-
target_data(realtime_system),
Diffs = m5.
*/
/* Difference 6- the importance of properties to the object movement.
   This rule is double strength in order to differentiate between similar
   OAP and OCP structures. */

difference_matches(Diffs,oap) :-
property_mapping(_,object,different_object_types,oap),
Diffs = m6.

difference_matches(Diffs,oap) :-
property_mapping(_,object,different_object_types,oap),
Diffs = m6.

difference_matches(Diffs,oap) :-
property_mapping(_,object,different_object_types,oap),
Diffs = m6.

difference_matches(Diffs,ocp) :-
property_mapping(_,object,resource,ocp),
Diffs = m6.

difference_matches(Diffs,ocp) :-
property_mapping(_,object,resource,ocp),
Diffs = m6.

difference_matches(Diffs,ocp) :-
property_mapping(_,object,resource,ocp),
Diffs = m6.

difference_matches(Diffs,ocp) :-
property_mapping(_,slot,resource_container,ocp),
Diffs = m6.

difference_matches(Diffs,ocp) :-
property_mapping(_,slot,resource_container,ocp),
Diffs = m6.

difference_matches(Diffs,ocp) :-
property_mapping(_,slot,resource_container,ocp),
Diffs = m6.

difference_matches(Diffs,omp) :-
object_matching(Tobj,object,omp),
not target_pdata(Tobj,_),
Diffs = m6.

difference_matches(Diffs,omp) :-
object_matching(Tobj,object,omp),
```

```
not target_pdata(Tobj,_),
Diffs = m6.

difference_matches(Diffs,omp) :-
object_matching(Tobj,object,omp),
not target_pdata(Tobj,_),
Diffs = m6.

/* Difference 7- Direction of movements between slots for OCP & OAP.
   These rules are also double strength, since it is important to
   discriminate between the OCP and OAP structures which have few
   structures. */

difference_matches(Diffs,ocp) :-
dynamic_mapping(_,_,_,object,slot,space2,move_many,_,ocp),
Diffs = m7.

difference_matches(Diffs,ocp) :-
dynamic_mapping(_,_,_,object,slot,space2,move_many,_,ocp),
Diffs = m7.

difference_matches(Diffs,oap) :-
dynamic_mapping(_,_,_,object,space,allocation,move_many,_,oap),
Diffs = m7.

difference_matches(Diffs,oap) :-
dynamic_mapping(_,_,_,object,space,allocation,move_many,_,oap),
Diffs = m7.

/* Difference 8- Difference to identify simple moves of the OPP
   structure, in particular to differentiate it from the OMP structure. */

difference_matches(Diffs,opp) :-
dynamic_mapping(_,_,_,object,space1,slot,move_one,_,opp),
Diffs = m8.

difference_matches(Diffs,opp) :-
dynamic_mapping(_,_,_,object,space1,slot,move_one,_,opp),
Diffs = m8.

difference_matches(Diffs,opp) :-
dynamic_mapping(_,_,_,object,slot,space2,move_one,_,opp),
Diffs = m8.

difference_matches(Diffs,opp) :-
dynamic_mapping(_,_,_,object,slot,space2,move_one,_,opp),
Diffs = m8.

/* Difference 9- Difference to identify simple moves of the OMP
   structure. */

difference_matches(Diffs,omp) :-
dynamic_mapping(_,_,_,object,slot,slot,move_one,_,omp),
Diffs = m9.

difference_matches(Diffs,omp) :-
dynamic_mapping(_,_,_,object,slot,slot,move_one,_,omp),
Diffs = m9.
```

```
difference_matches(Diffs,omp) :-
dynamic_mapping(_,_,_,object,slot,slot,move_one,_,omp),
Diffs = m9.

/* Differences based on the functionality of the problem. */

difference_matches(Diffs,ocp) :-
function_mapping(F,ocp),
Diffs = m9a.

difference_matches(Diffs,omp) :-
function_mapping(F,omp),
Diffs = m9a.

difference_matches(Diffs,oap) :-
function_mapping(F,oap),
Diffs = m9a.

difference_matches(Diffs,opp) :-
function_mapping(F,opp),
Diffs = m9a.

/* Differences between the OCPAA and OCPAB structures. */

/* Difference 1- important difference on the movements into the slot. */

difference_matches(Diffs,ocpaa) :-
dynamic_mapping(_,_,_,object,space1,slot,move_many,Score,ocpaa),
Diffs = m10.

difference_matches(Diffs,ocpaa) :-
dynamic_mapping(_,_,_,object,space1,slot,move_many,Score,ocpaa),
Diffs = m10.

difference_matches(Diffs,ocpaa) :-
dynamic_mapping(_,_,_,object,space1,slot,move_many,Score,ocpaa),
Diffs = m10.

difference_matches(Diffs,ocpaa) :-
dynamic_mapping(_,_,_,object,space1,slot,move_many,Score,ocpaa),
Diffs = m10.

difference_matches(Diffs,ocpab) :-
dynamic_mapping(_,_,_,object,space2,slot,move_many,Score,ocpab),
Diffs = m10.

difference_matches(Diffs,ocpab) :-
dynamic_mapping(_,_,_,object,space2,slot,move_many,Score,ocpab),
Diffs = m10.

difference_matches(Diffs,ocpab) :-
dynamic_mapping(_,_,_,object,space2,slot,move_many,Score,ocpab),
Diffs = m10.

/* Difference 2- difference between the requirements for two problems. */

difference_matches(Diffs,ocpaa) :-
calc_reqt2(_,_,slot,object,contains_many,minimum_qty,ocpaa),
Diffs = m11.
```

```
difference_matches(Diffs,ocpab) :-
calc_reqt2(_,_,space2,object,contains_many,date_limit,ocpab),
Diffs = m11.

/* Difference 3- difference on functionality of state transitions. */

difference_matches(Diffs,ocpab) :-
function_mapping(F,ocpab),
Diffs = m11a.

difference_matches(Diffs,ocpaa) :-
function_mapping(F,ocpaa),
Diffs = m11a.

/* Differences between the OCPBA & BB structures. */

/* Difference 1- the structure of objects within the major slot. */

difference_matches(Diffs,ocpba) :-
static_mapping(_,_,slot,smallslot,has_many,Score,ocpba),
Diffs = m12.

difference_matches(Diffs,ocpbb) :-
static_mapping(_,_,slot,object,contains_many,Score,ocpbb),
Diffs = m12.

/* Difference 2- the space of object movement out of the slot. */

difference_matches(Diffs,ocpbb) :-
target_scope(_,Obj1,Obj2,Obj3,move_many),
object_matching(Obj1,object,_),
object_matching(Obj2,slot,_),
object_matching(Obj3,space2,_),
Diffs = m13.

/* Difference 3 - the functional mapping on state transformations. */

difference_matches(Diffs,ocpbb) :-
function_mapping(F,ocpbb),
Diffs = m13a.

difference_matches(Diffs,ocpba) :-
function_mapping(F,ocpba),
Diffs = m13a.
```

```
/* These are the rules to match the labels assigned to each acp
   to assist the analogous matching process */

/* First layer of rules to identify the category of label match which has
   occured */

perfect_label(Selected_acp) :-
calc_label(Selected_acp,Label_score),
Label_score =:= 3.

good_label(Selected_acp) :-
calc_label(Selected_acp,Label_score),
Label_score =:= 2.

poor_label(Selected_acp) :-
calc_label(Selected_acp,Label_score),
Label_score =:= 1.

fail_label(Selected_acp) :-
calc_label(Selected_acp,Label_score),
Label_score =:= 0.

/* Second layer to generate a list of matched labels then count the list */

calc_label(Selected_acp,Label_score) :-
setof(Acp_label,labels_match(Acp_label,Selected_acp),Labels_list),
length(Labels_list,Label_score).

calc_label(Selected_acp,Label_score) :-
not labels_match(Acp_label,Selected_acp),
Label_score is 0.

/* Third layer checks each acp-label for membership of the target labels
   list */

labels_match(Acp_label,Selected_acp) :-
acp_label(Acp_label,Selected_acp),
target_label(Target_label),
compare(=,Acp_label,Target_label).
```

```
/* Program to infer the analogical existence of addition domain facts
  - this program only checks the static and dynamic knowledge
  mappings since they identify all possible relations linking mapped
  objects */

create_data(Tobj1,Tobj2,Trelation) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Acp).

create_data(Tobj1,Tobj2,Tobj3,Srelation) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Trelation,Acp).
```

```
/* Program to examine the match with non-critical physical aspects of
   a good-matching acp, in order to further justify and extend the
   analogy */

/* The first rule identifies basic match with the physical structure of the
   acp */

physical_acp(Acp) :-
findall(Prop,prop_phymatch(_,_,Prop,Acp),List),
length(List,Total_matches),
acp_phymappings(Acp,All_matches),
Total_matches/All_matches >= 0.65.

/* Second level rules to actually match physical features. A match only
   occurs if there is an object match identified in the recorded mappings,
   so need to develop this component. */

prop_phymatch(Tobj,Sobj,Tproperty,Acp) :-
target_phyprop(Tobj,Tproperty),
acp_phyprop(Sobj,Sproperty,Acp),
compare(=,Tproperty,Sproperty),
rec_objectmatch(Tobj,Sobj,_,_).
```

```
/* This program identifies confirmed analogous mappings and records
   these mappings in the data base in order to use for explanation and
   support of the analogy. Ira only records successful analogies - i.e. those
   which are a good analogous match. The program also records the name
   of the matched acp. There is a findall to ensure that all relevant and
   good mappings with an ACP are recorded. */

record_acpmatch(Acp) :-
findall(Acp,record_match(Acp),Anylist).

record_match(Acp) :-
enable_item('Control','Identify Mappings'),
assertz(rec_acpmatch(Acp)).

record_match(Acp) :-
static_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Score,Acp),
assertz(rec_statmapping(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Score,Acp)).

record_match(Acp) :-
dynamic_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Trelation,
Score,Acp),
assertz(rec_dynmapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,
Trelation,Score,Acp)).

record_match(Acp) :-
property_mapping(Tobj1,Sobj1,Sproperty,Acp),
assertz(rec_propmapping(Tobj1,Sobj1,Sproperty,Acp)).

record_match(Acp) :-
condition_mapping(Tobj1,Tobj2,Tobj3,Sobj1,
Sobj2,Sobj3,Rel,Condition,Acp),
assertz(rec_condmapping(Tobj1,Tobj2,Tobj3,Sobj1,
Sobj2,Sobj3,Rel,Condition,Acp)).

record_match(Acp) :-
function_mapping(Amvmt,Acp),
assertz(rec_funcmapping(Amvmt,Acp)).

record_match(Acp) :-
calc_reqt1(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Acp),
assertz(rec_reqtmapping1(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Acp)).

record_match(Acp) :-
calc_reqt2(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Ident,Acp),
assertz(rec_reqtmapping2(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Ident,Acp)).

record_match(Acp) :-
calc_scope(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Acp),
assertz(rec_scopemapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Acp)).

record_match(Acp) :-
labels_match(Label,Acp),
assertz(rec_labelmapping(Label,Acp)).

record_match(Acp) :-
prop_phymatch(Tobj,Sobj,Property,Acp),
assertz(rec_phymapping(Tobj,Sobj,Property,Acp)).

/* Set of rules to record all object matches upon completion of a match.
```

These rules are similar to those which operate on current acp, but to avoid unnecessary complexities it is necessary to develop and identify them separately. A FINDALL rule is used to ensure all of the necessary assertions are achieved. These call rules which identify specific matchings which examine the recorded predicates determined above. The first, high-level rule is the FINDALL rule which operates two searches. */

```
record_match(Acp) :-
findall(Acp,record_oldmapping(Acp),Otherlist),
findall(Acp,record_newmapping(Acp),Anylist).

/* The second-level ruleset which identifies each object-matching pair.
   There are two rules here. The first updates existing object matches
   while the second creates new object matches which did not exist. The
   critical control in this mechanism is recording the ACP in the object
   matching. The firing loop in each of these rules in determined by the
   ACP label in the match - do not fire in the object match has been
   altered to the current ACP. */

record_oldmapping(Acp) :-
objrec_match(Tobj,Sobj,_,Acp),
rec_objectmatch(Tobj,Sobj,Oldscore,Oldacp),
Acp=\=Oldacp,change_objectmatch(Tobj,Sobj,Oldscore,Acp).

change_objectmatch(Tobj,Sobj,Oldscore,Acp) :-
current_matches(Tobj,Sobj,Score,Acp),
Newscore is Oldscore + Score,
retract(rec_objectmatch(Tobj,Sobj,Oldscore,_)),
assertz(rec_objectmatch(Tobj,Sobj,Newscore,Acp)),!.

record_newmapping(Acp) :-
objrec_match(Tobj,Sobj,_,Acp),
not rec_objectmatch(Tobj,Sobj,_,_),
current_matches(Tobj,Sobj,Score,Acp),
assertz(rec_objectmatch(Tobj,Sobj,Score,Acp)).

/* The third-level ruleset which identifies specific recorded matchings,
   from those which were previously recorded for the acp, or those
   identified during mapping with previous acps. */

objrec_match(Tobj,Sobj,Score,Acp) :-
rec_statmapping(Tobj,_,Sobj,_,_,Score,Acp).

objrec_match(Tobj,Sobj,Score,Acp) :-
rec_statmapping(_,Tobj,_,Sobj,_,Score,Acp).

objrec_match(Tobj,Sobj,Score,Acp) :-
rec_dynmapping(Tobj,_,_,Sobj,_,_,_,Score,Acp).

objrec_match(Tobj,Sobj,Score,Acp) :-
rec_dynmapping(_,Tobj,_,_,Sobj,_,_,Score,Acp).

objrec_match(Tobj,Sobj,Score,Acp) :-
rec_dynmapping(_,_,Tobj,_,_,Sobj,_,Score,Acp).

/* Rule to determine object scores for the match with the current acp
   only. This stage in the program must only identify the score for the
   current acp matches, so not to cummulate the score twice. It is a
```

separate routine based on that used in the static and dynamic-mapping routines. It repeats the same techniques, and omits to count the previous other object totals, so large numbers of the processes are matched. */

```
/* Basic rule for identifying all possible current object matches. */

current_matches(Tobj,Sobj,Totalscore,Acp) :-
findall(Score,current_match(Tobj,Sobj,Score,Acp),Scorelist),
object_scores(Scorelist,0,Totalscore).

current_match(Tobj,Sobj,Score,Acp) :-
current_static(Tobj,_,Sobj,_,_,Score,Acp).

current_match(Tobj,Sobj,Score,Acp) :-
current_static(_,Tobj,_,Sobj,_,Score,Acp).

current_match(Tobj,Sobj,Score,Acp) :-
current_dynamic(Tobj,_,_,Sobj,_,_,_,Score,Acp).

current_match(Tobj,Sobj,Score,Acp) :-
current_dynamic(_,Tobj,_,_,Sobj,_,_,Score,Acp).

current_match(Tobj,Sobj,Score,Acp) :-
current_dynamic(_,_,Tobj,_,_,Sobj,_,Score,Acp).

/* Matching techniques borrowed from structure mapping program. It
   remains similar expect the routine names called from above. One
   result from using this technique is that objects score double in the
   event of a tie, so leave this in the system for the moment, although
   may need fixing at a later date. */

/* Four basic matching rules called by the above techniques. */

current_static(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Score,Acp) :-
acp_sdata(Sobj1,Sobj2,Srelation,Acp),
target_sdesc(Tobj1,Tobj2,Trelation),
compare(=,Srelation,Trelation),
not rec_statmapping(T1,T2,Sobj1,Sobj2,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
structure_scurrent(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Score,Acp),
possible_scurrents(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Scores,Acp),
sort(Scores,Sorted_scores,[],1),
Sorted_scores = [Other_score|Rest],
Score >= Other_score.

current_static(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Score,Acp) :-
acp_sdata(Sobj1,Sobj2,Srelation,Acp),
target_sdesc(Tobj1,Tobj2,Trelation),
compare(=,Srelation,Trelation),
not rec_statmapping(T1,T2,Sobj1,Sobj2,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
not multi_stmapping(Trelation,Acp),
structure_scurrent(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Score,Acp).

current_dynamic(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,
Trelation,Score,Acp) :-
```

```
acp_ddata(_,Sobj1,Sobj2,Sobj3,Srelation,Acp),
target_ddesc(_,Tobj1,Tobj2,Tobj3,Trelation),
compare(=,Srelation,Trelation),
not rec_dynmapping(T1,T2,T3,Sobj1,Sobj2,Sobj3,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
not detailed_object(Tobj3,Acp),
structure_dcurrent(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Score,Acp),
possible_dcurrents(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Scores,Acp),
sort(Scores,Sorted_scores,[],1),
Sorted_scores = [Other_score|Rest],
Score >= Other_score.

current_dynamic(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,
Trelation,Score,Acp) :-
acp_ddata(_,Sobj1,Sobj2,Sobj3,Srelation,Acp),
target_ddesc(_,Tobj1,Tobj2,Tobj3,Trelation),
compare(=,Srelation,Trelation),
not rec_dynmapping(T1,T2,T3,Sobj1,Sobj2,Sobj3,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
not detailed_object(Tobj3,Acp),
not multi_dymapping(Trelation,Acp),
structure_dcurrent(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Score,Acp).

/* Rules to match the candidate structure. */

structure_scurrent(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Score,Acp) :-
findall(Rel,object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Acp),
Slist),length(Slist,Score).

structure_dcurrent(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Score,Acp) :-
findall(Rel,object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Acp),
Dlist),length(Dlist,Score).

/* Rules to identify the alternative options required for consideration. */

possible_scurrents(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Scorelist,Acp) :-
findall(Score,(
target_sdesc(Toth1,Toth2,Rel),
possible_scurrent(Sobj1,Sobj2,Toth1,Toth2,Rel,Score,Acp),
not same_staticobjects(Tobj1,Tobj2,Toth1,Toth2)),
Scorelist),!.

possible_scurrent(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Score,Selected_acp) :-
target_sdesc(Tobj1,Tobj2,Rel),
findall(Rel,object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp),Slist),
length(Slist,Score).

possible_dcurrents(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Slist,Acp) :-
findall(Score,(
target_ddesc(_,Toth1,Toth2,Toth3,Rel),
possible_dcurrent(Sobj1,Sobj2,Sobj3,Toth1,Toth2,Toth3,Rel,Score,Acp),
not same_dynamicobjects(Tobj1,Tobj2,Tobj3,Toth1,Toth2,Toth3)),
Slist),!.
```

```
possible_dcurrent(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Score,Acp) :-
target_ddesc(_,Tobj1,Tobj2,Tobj3,Rel),
findall(Rel,object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Acp),
Dlist),length(Dlist,Score).
```

```
/* Program to investigate matching between requirements for a selected
   acp - as with scoping the requirements features are quite simplistic,
   and the program is directly linked to the features of each acp */

/* First layer of rules to determine the level of requirements matching */

perfect_requirements(Selected_acp) :-
calc_requirements(Selected_acp,Reqts_score),
Reqts_score > 75.

good_requirements(Selected_acp) :-
calc_requirements(Selected_acp,Reqts_score),
Reqts_score =< 75,
Reqts_score > 50.

poor_requirements(Selected_acp) :-
calc_requirements(Selected_acp,Reqts_score),
Reqts_score =< 50,
Reqts_score > 25.

fail_requirements(Selected_acp) :-
calc_requirements(Selected_acp,Reqts_score),
Reqts_score =< 25,!.

/* Second layer of rules to identify the Requirements score for
   each acp. Two rules fire, one to get either type of requirement
   which may exist in the system. An initial version of the rule is needed
   to allow for the possibility of no available requirements. */

calc_requirements(Selected_acp,100) :-
acp_reqt_total(Selected_acp,0),!.

calc_requirements(Selected_acp,Reqts_score) :-
get_reqt1(Selected_acp,Score1),
get_reqt2(Selected_acp,Score2),
Total = Score1 + Score2,
acp_reqt_total(Selected_acp,Acp_total),
Reqts_score is (Total/Acp_total) * 100.

/* Get_reqt rules for each requirement type give a score for the total
   number of matches which occur for both types of requirement. This
   program counts up the instances pf matches, then gets rid of
   duplicates to give the true number of matches for the program */

get_reqt1(Selected_acp,Score) :-
findall(calc_reqt1(_,_,_,_,_,Selected_acp),L1),
length(L1,Score).

get_reqt2(Selected_acp,Score) :-
findall(calc_reqt2(_,_,_,_,_,Selected_acp),L1),
length(L1,Score).

/* Third layer of rules to identify individual instances of matches with
   the requirements - three rules exist to cater for the possible
   descriptions of the requirements in the system. Note that the target
   description is the reformulated description, so that the target may
   successfully match as possible. */

calc_reqt1(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Selected_acp) :-
```

```
acp_reqt(Sobj1,Sobj2,Relation,Selected_acp),
target_reqts(Tobj1,Tobj2,Relation),
static_mapped(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Score,Selected_acp).

calc_reqt2(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Identifier,Selected_acp) :-
acp_reqt(Sobj1,Sobj2,Relation,Identifier,Selected_acp),
target_reqts(Tobj1,Tobj2,Relation,Identifier),
static_mapped(Tobj1,Tobj2,Sobj1,Sobj2,Relation,Score,Selected_acp).
```

```
/* Programs to determine the scope details of each selected acp -
   due to the limited extent played by problem scope, these rules are
   purely acp-dependent, and directly identify the match_scope value
   without the need for a calc_scope to be done. */

perfect_scope(Selected_acp) :-
calc_scope(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Relation,Selected_acp).

calc_scope(Tobj1,Tobj2,Tobj3,object,slot,space2,move_many,ocp) :-
acp_scope(Atran,ocp),
acp_ddata(Atran,object,slot,space2,move_many,ocp),
target_scope(Ttran),
target_ddesc(Ttran,Tobj1,Tobj2,Tobj3,move_many),
dynamic_mapping(Tobj1,Tobj2,Tobj3,
object,slot,space2,move_many,_,ocp).

calc_scope(Tobj1,Tobj2,Tobj3,object,space2,slot,move_many,ocpab) :-
acp_scope(Atran,ocpab),
acp_ddata(Atran,object,space2,slot,move_many,ocpab),
target_scope(Ttran),
target_ddesc(Ttran,Tobj1,Tobj2,Tobj3,move_many),
dynamic_mapping(Tobj1,Tobj2,Tobj3,
object,space2,slot,move_many,_,ocpab).

calc_scope(Tobj1,Tobj2,Tobj3,object,space1,slot,move_many,ocpbb) :-
acp_scope(Atran,ocpbb),
acp_ddata(Atran,object,space1,slot,move_many,ocpaa),
target_scope(Ttran),
target_ddesc(Ttran,Tobj1,Tobj2,Tobj3,move_many),
rec_dynmapping(Tobj1,Tobj2,Tobj3,
object,space1,slot,move_many,_,ocpaa).

calc_scope(Tobj1,Tobj2,Tobj3,object,slot,slot,move_one,omp) :-
acp_scope(Atran,omp),
acp_ddata(Atran,object,slot,slot,move_one,omp),
target_scope(Ttran),
target_ddesc(Ttran,Tobj1,Tobj2,Tobj3,move_one),
dynamic_mapping(Tobj1,Tobj2,Tobj3,object,slot,slot,move_one,_,omp).

calc_scope(Tobj1,Tobj2,Tobj3,object,space1,slot,move_one,opp) :-
acp_scope(Atran,opp),
acp_ddata(Atran,object,space1,slot,move_one,opp),
target_scope(Ttran),
target_ddesc(Ttran,Tobj1,Tobj2,Tobj3,move_one),
dynamic_mapping(Tobj1,Tobj2,Tobj3,object,space1,slot,move_one,_,opp).

calc_scope(Tobj1,Tobj2,Tobj3,object,slot,space2,move_one,opp) :-
acp_scope(Atran,opp),
acp_ddata(Atran,object,slot,space2,move_one,opp),
target_scope(Ttran),
target_ddesc(Ttran,Tobj1,Tobj2,Tobj3,move_one),
dynamic_mapping(Tobj1,Tobj2,Tobj3,object,slot,space2,move_one,_,opp).

good_scope(Selected_acp) :- !,fail.
poor_scope(Selected_acp) :- !,fail.
fail_scope(Selected_acp) :- !,fail.
```

```
/* There are two major sequential goals to the selection control program.
   The first goal (checking) works down to identify the appropriate
   additional processing which is necessary to identify final good match, &
   provide additional interaction with analyst. Second sequential goal
   (matching) works from the new data about problem, and either searches
   then search space or accepts to agreed input from the analyst (requiring
   no further processing. These goals are distributed about the processing
   of the search mechanism. Here we identify goal for first-pass search of
   of the space, and the goal which researches the space once the analyst
   has been prompted, and the engine has received guidance from the
   analyst. */

/* The first rule (check a new sub-tree in the search space) checks for
   correct father and searches for matches with the existing data. It calls
   a second rule to do summarisation of good and partial searches. It
   is also important to check for bottom of the search tree, i.e. no
   candidate sons to be searched, so a check is made initially in the
   rule. The second version of this rule is included to stop the search if
   no system name has been input - this is obligatory for the system to
   run. */

searching_acps(Prev_acp) :-
not father(Prev_acp,_),
stop_searching(Prev_acp),!.

searching_acps(Prev_acp) :-
not target_name(Name),
beep(60),mdialog(85,130,140,350,
[button(100,127,20,100,'Continue'),
text(20,20,64,310,'You must identify the name of the system before attempting an analogous match. Please
CONTINUE then use the OTHER INPUTS menu to identify the name of the system.')],Btn),!.

searching_acps(Prev_acp) :-
target_name(Name),
banner(acp_checking(Prev_acp),['Please be patient - Ira is reasoning analogously to match
the',Name,'problem'],150,110).

/* Second-level search rules which react to results from search and
   if necessary elicit additional knowledge from the analyst. */

/* One good match - the easiest case. Record ACP match and search at
   next level. */

acp_checking(Prev_acp) :-
good_match(Prev_acp,Glist),
length(Glist,T),T = 1,
Glist = [New_acp|None],
record_acpmatch(New_acp),
searching_acps(New_acp),!.

/* Two good matches - analyst decision between possible options */

acp_checking(Prev_acp) :-
good_match(Prev_acp,Glist),
length(Glist,T), T >= 1,
goodmatches_dialogue(Glist),!.

/* One partial match, eliciting further target knowledge then search */
```

```
acp_checking(Prev_acp) :-
partial_match(Prev_acp,Plist),
length(Plist,T),T = 1,
Plist = [Acp|None],
partmatch1_dialogue(Acp),!.

/* Select between several partial matches presented to analysts */

acp_checking(Prev_acp) :-
partial_match(Prev_acp,Plist),
length(Plist,T),T >= 1,
partmatch2_dialogue(Plist),!.

/* Failed match - stop searching */

acp_checking(Prev_acp) :-
stop_searching(Prev_acp).

/* First-level rule to research problem space once additional
   knowledge elicited from analyst. */

matching_acps(Prev_acp) :-
target_name(Name),
banner(acp_matching(Prev_acp),['Please be patient - Ira is reasoning analogously to match
the',Name,'problem'],150,110).

/* Second level rules to identify results of searching space after
   additional knowledge elicited from analyst. There are fewer routines
   here than for the first pass checking-acp due to the possible
   nature of the knowledge. */

/* Single good match - the easiest case. Record the matched ACP and
   search at the next level of hierarchy. */

acp_matching(Prev_acp) :-
good_match(Prev_acp,Glist),
length(Glist,T),T = 1,
Glist = [New_acp|None],
record_acpmatch(New_acp),
searching_acps(New_acp),!.

/* Two good matches - analyst decision between possible options */

acp_matching(Prev_acp) :-
good_match(Prev_acp,Glist),
length(Glist,T), T >= 1,
goodmatches_dialogue(Glist),!.

/* One partial match - unable to extend searching any further. */

acp_matching(Prev_acp) :-
partial_match(Prev_acp,Plist),
length(Plist,T),T = 1,
Plist = [New_acp|None],
acceptmatches_dialogue(Plist),!.

/* Several partial matches - unable to extend searching any further. */

acp_matching(Prev_acp) :-
```

```
partial_match(Prev_acp,Plist),
length(Plist,T),T >= 1,
acceptmatches_dialogue(Plist),!.

/* Failed match, due to much changed target domain data. */

acp_matching(Prev_acp) :-
stop_searching(Prev_acp).
```

```
/* This program is central to identifying the analogy with an acp. It
   determines analogous mappings between object-relations, and checks
   the validity of each relation mapping against all other associated
   mappings to ensure a coherent structure to the analogy */

/* Rules to map object-relations for static structural knowledge. There
   are two rules - the first is more complex and decides between
   'Relations' which have several possible sets of object mappings which
   must be added up and compared. The system maps the objects
   around the relation which best fit into the remainder of the analogy
   structure. There are two separate rules which are needed: i) to
   ensure the program does not compare identical analogous mappings,
   & ii) to check for the existence of several possible mappings in an
   acp to decide which version of the mapping rule is required. If two
   object mappings have equal best fit in the structure, then both are
   analogously-mapped. */

static_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Score,Acp) :-
acp_sdata(Sobj1,Sobj2,Srelation,Acp),
target_sdesc(Tobj1,Tobj2,Trelation),
compare(=,Srelation,Trelation),
not rec_statmapping(T1,T2,Sobj1,Sobj2,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
static_structure(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Score,Acp),
static_possibles(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Scores,Acp),
sort(Scores,Sorted_scores,[],1),
Sorted_scores = [Other_score|Rest],
Score >= Other_score.

static_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Score,Acp) :-
acp_sdata(Sobj1,Sobj2,Srelation,Acp),
target_sdesc(Tobj1,Tobj2,Trelation),
compare(=,Srelation,Trelation),
not rec_statmapping(T1,T2,Sobj1,Sobj2,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
not multi_stmapping(Trelation,Acp),
static_structure(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Score,Acp).

/* Check rule to support the above two analogous mappings */

multi_stmapping(Relation,Acp) :-
findall(Relation,target_sdesc(_,_,Relation),Tlist),
findall(Relation,acp_sdata(_,_,Relation,Acp),Slist),
length(Tlist,T),
length(Slist,S),
T + S > 2.

/* Rules to map object-relation for dynamic structural knowledge. There
   are two rules - the first is more complex and decides between
   'Relations' which have several possible sets of object mappings which
   must be added up and compared. The system maps the objects
   around the relation which best fit into the remainder of the analogy
   structure. There are two separate rules which are needed: i) to
   ensure the program does not compare identical analogous mappings,
   & ii) to check for the existence of several possible mappings in an
   acp to decide which version of the mapping rule is required. */
```

```
dynamic_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,
Trelation,Score,Acp) :-
acp_ddata(_,Sobj1,Sobj2,Sobj3,Srelation,Acp),
target_ddesc(_,Tobj1,Tobj2,Tobj3,Trelation),
compare(=,Srelation,Trelation),
not rec_dynmapping(T1,T2,T3,Sobj1,Sobj2,Sobj3,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
not detailed_object(Tobj3,Acp),
dynamic_structure(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Score,Acp),
dynamic_possibles(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Scores,Acp),
sort(Scores,Sorted_scores,[],1),
Sorted_scores = [Other_score|Rest],
Score >= Other_score.

dynamic_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,
Trelation,Score,Acp) :-
acp_ddata(_,Sobj1,Sobj2,Sobj3,Srelation,Acp),
target_ddesc(_,Tobj1,Tobj2,Tobj3,Trelation),
compare(=,Srelation,Trelation),
not rec_dynmapping(T1,T2,T3,Sobj1,Sobj2,Sobj3,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
not detailed_object(Tobj3,Acp),
not multi_dymapping(Trelation,Acp),
dynamic_structure(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Score,Acp).

/* Check rule to support the above two analogous mappings */

multi_dymapping(Relation,Acp) :-
findall(Relation,target_ddesc(_,_,_,_,Relation),Tlist),
findall(Relation,acp_ddata(_,_,_,_,Relation,Acp),Slist),
length(Tlist,T),
length(Slist,S),
T + S > 2.

/* Rules to map objects which have analogous property features */

property_mapping(Tobj1,Sobj1,Sproperty,Acp) :-
acp_pdata(Acp,Sobj1,Sproperty),
target_pdata(Tobj1,Tproperty),
compare(=,Sproperty,Tproperty),
not detailed_object(Tobj1,Acp),
object_matching(Tobj,Sobj,Acp),
not rec_propmapping(T,Sobj1,Sproperty,_).

/* Rules to map conditions which control the movement of objects */

condition_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,
Rel,Condition,Acp) :-
acp_cdata(Amvmt,Condition,Acp),
acp_ddata(Amvmt,Sobj1,Sobj2,Sobj3,Rel,Acp),
target_cdata(Tmvmt,Condition),
target_ddata(Tmvmt,Tobj1,Tobj2,Tobj3,Rel),
dynamic_mapped(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Score,Acp),
not rec_condmapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,
```

```
Sobj3,Rel,Condition,_).

/* Sub-program to identify the number of all identified analogous
   mappings with neighbouring relations identified by the connecting
   mappings - i.e. testing the extent of analogous match around the
   currently validated analogous matching - the programs static_
   structure and dynamic_structure are incorporated into the original
   matching rules, depending upon the original static or dynamic mapping.
   For each rule type there are two instances: (i) assumption that recorded
   mappings exist, retrieved by a rule called object totals, (ii) no object
   mappings prevelant to that mapping. */

static_structure(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Score,Selected_acp) :-
findall(Rel,object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp),
Slist),length(Slist,Score1),
object_total1(Sobj1,Sobj2,Tobj1,Tobj2,Scorelist),
object_scores(Scorelist,0,Score2),
Score is Score1 + Score2,!.

static_structure(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Score,Selected_acp) :-
findall(Rel,object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp),
Slist),length(Slist,Score).

dynamic_structure(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Score,Acp) :-
findall(Rel,object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Acp),
Dlist),length(Dlist,Score1),
object_total2(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Scorelist),
object_scores(Scorelist,0,Score2),
Score is Score1 + Score2,!.

dynamic_structure(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Score,Acp) :-
findall(Rel,object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Acp),
Dlist),length(Dlist,Score).

/* Corresponding sub-program to get other target mappings which might
   map to the abstract acp relation being mapped. It is more complex
   than the above rules, in that it has to retrieve a single list of all scores
   from other, different mappings, which are then passed to the main
   program for comparing. The top-rule (possiblES) obtains the list of
   scores for all object-mappings with same relation except that being
   matched in the main program (hence not-same rule). The lower rule
   (possiblE) passes takes each alternative option, and counts number of
   mappings to check whether it is the highest score for the relation or
   not. Note that in case of empty set (no possibles) scorelist should be
   set to 0 to avoid higher-level rules. This is achieved by additional rule
   included at static_possibleS level to identify no static_possible matches.
   */

/* Static mapping possibles (two rules exist). The first rule runs when
   there is no empty set, the second rule defaults to required empty set. */

static_possibles(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Scorelist,Acp) :-
findall(Score,(
target_sdesc(Toth1,Toth2,Rel),
static_possible(Sobj1,Sobj2,Toth1,Toth2,Rel,Score,Acp),
not same_staticobjects(Tobj1,Tobj2,Toth1,Toth2)),
Scorelist),!.

static_possible(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Score,Selected_acp) :-
```

```
target_sdesc(Tobj1,Tobj2,Rel),
findall(Rel,object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp),Slist),
length(Slist,Score1),
object_total1(Sobj1,Sobj2,Tobj1,Tobj2,Scorelist),
object_scores(Scorelist,0,Score2),
Score is Score1 + Score2,!.

static_possible(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Score,Selected_acp) :-
target_sdesc(Tobj1,Tobj2,Rel),
findall(Rel,object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp),Slist),
length(Slist,Score).

/* Dynamic mapping possibles (two rules exist as for statics). */

dynamic_possibles(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Slist,Acp) :-
findall(Score,(
target_ddesc(_,Toth1,Toth2,Toth3,Rel),
dynamic_possible(Sobj1,Sobj2,Sobj3,Toth1,Toth2,Toth3,Rel,Score,Acp),
not same_dynamicobjects(Tobj1,Tobj2,Tobj3,Toth1,Toth2,Toth3)),
Slist),!.

dynamic_possible(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Score,Acp) :-
target_ddesc(_,Tobj1,Tobj2,Tobj3,Rel),
findall(Rel,object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Acp),
Dlist),length(Dlist,Score1),
object_total2(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Scorelist),
object_scores(Scorelist,0,Score2),
Score is Score1 + Score2,!.

dynamic_possible(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Score,Acp) :-
target_ddesc(_,Tobj1,Tobj2,Tobj3,Rel),
findall(Rel,object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Acp),
Dlist),length(Dlist,Score).

/* Check rules to support the static_ and dynamic_possible programs. */

same_staticobjects(Tobj1,Tobj2,Toth1,Toth2) :-
compare(=,Tobj1,Toth1),
compare(=,Tobj2,Toth2).

same_dynamicobjects(Tobj1,Tobj2,Tobj3,Toth1,Toth2,Toth3) :-
compare(=,Tobj1,Toth1),
compare(=,Tobj2,Toth2),
compare(=,Tobj3,Toth3).

/* Third level of rules to assimilate previous object mappings during ACP
   mapping. */

object_total1(Sobj1,Sobj2,Tobj1,Tobj2,Scorelist) :-
findall(Score,rec_objectmatch(Tobj1,Sobj1,Score,_),List1),
findall(Score,rec_objectmatch(Tobj2,Sobj2,Score,_),List2),
append(List1,List2,Scorelist).

object_total2(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Scorelist) :-
findall(Score,rec_objectmatch(Tobj1,Sobj1,Score,_),List1),
findall(Score,rec_objectmatch(Tobj2,Sobj2,Score,_),List2),
findall(Score,rec_objectmatch(Tobj3,Sobj3,Score,_),List3),
append(List1,List2,List12),
append(List12,List3,Scorelist).
```

```
/* Additional rule to calculate a total object score from a list of object
   scores. Based on Shapiro's book pp 129. */

object_scores([Score|Scores],T1,Sumscore) :-
T2 is T1+Score,
object_scores(Scores,T2,Sumscore).
object_scores([],Sumscore,Sumscore).

/* Third level of rules which identify the instances when two objects
   correctly correspond to the candidate set of objects.
   This set of rules are quite numerous and complex for several
   reasons. The rules call static- and dynamic-mapping rules with
   different names so that prolog is able to terminate. Several aspects
   of these rules make them complex and numerous:
   i) need different rules to check a static-mapping versus a dynamic-
      mapping - they must be redefined here so that these rules have
      the appropriate variables to reason with,
   ii) each static and dynamic rule must be checked against several
      combinations of neighbouring static and dynamic rules, and
   iii) neighbouring mappings calculated in the current acp against
      those determined from recorded mappings in the previously
      matched acps.

   These sections are broken down with comments to make these
   rules are more obvious */

/* static_mapping, same acp */

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
stat_mapping(Tnew,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
stat_mapping(_,Tnew,_,Snew,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
dyn_mapping(Tnew,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
dyn_mapping(Tnew,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
dyn_mapping(_,Tnew,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).
```

```
object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
dyn_mapping(_,Tnew,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
dyn_mapping(_,_,Tnew,_,_,Snew,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
dyn_mapping(_,_,Tnew,_,_,Snew,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
prop_mapping(Tnew,Snew,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
prop_mapping(Tnew,Snew,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).
```

```
object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_stmap(Sobj1,Sobj2,Tobj1,Tobj2,Rel,Selected_acp) :-
stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Selected_acp),
cond_mapping(_,_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

/* Dynamic mapping, same acp */

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
stat_mapping(Tnew,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
stat_mapping(Tnew,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
stat_mapping(Tnew,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
stat_mapping(_,Tnew,_,Snew,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
```

```
stat_mapping(_,Tnew,_,Snew,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
stat_mapping(_,Tnew,_,Snew,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(Tnew,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(_,Tnew,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(_,_,Tnew,_,_,Snew_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(Tnew,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(_,Tnew,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(_,_,Tnew,_,_,Snew_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(Tnew,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(_,Tnew,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
```

```
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
dyn_mapping(_,_,Tnew,_,_,Snew_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
prop_mapping(Tnew,Snew_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
prop_mapping(Tnew,Snew_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
prop_mapping(Tnew,Snew_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj1),
compare(=,Tnew,Tobj1).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).
```

```
object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj2),
compare(=,Tnew,Tobj2).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

object_dymap(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,Rel,Selected_acp) :-
dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Selected_acp),
cond_mapping(_,_,_,_,_,Tnew,_,_,_,_,_,_,Snew,_,_,Selected_acp),
compare(=,Snew,Sobj3),
compare(=,Tnew,Tobj3).

/* Fourth level identifying stat-mapping, dyn-mapping, prop-mapping
   and cond-mapping, the basic mapping programs to identify isloated
   chunks of matching structure. */

stat_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Trelation,Acp) :-
```

```
acp_sdata(Sobj1,Sobj2,Srelation,Acp),
target_sdesc(Tobj1,Tobj2,Trelation),
compare(=,Srelation,Trelation).

dyn_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Trelation,Acp) :-
acp_ddata(_,Sobj1,Sobj2,Sobj3,Srelation,Acp),
target_ddesc(_,Tobj1,Tobj2,Tobj3,Trelation),
compare(=,Srelation,Trelation).

prop_mapping(Tobj,Sobj,Property,Acp) :-
acp_pdata(Acp,Sobj,Property),
target_pdata(Tobj,Property).

cond_mapping(Tobj1,Tobj2,Tobj3,Rel1,Tobj4,Tobj5,Rel2,
Sobj1,Sobj2,Sobj3,Rel1,Sobj4,Sobj5,Rel2,Condition,Acp) :-
acp_cdata(Tobj1,Tobj2,Tobj3,Rel1,Tobj4,Tobj5,Rel2,Condition,Acp),
target_cdata(Sobj1,Sobj2,Sobj3,Rel1,Sobj4,Sobj5,Rel2,Condition).

/* Subroutines to identify static and dynamic mappings which occured
   during either the current or previous acps. */

static_mapped(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Score,Acp) :-
static_mapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Score,Acp).

static_mapped(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Score,Acp) :-
rec_statmapping(Tobj1,Tobj2,Sobj1,Sobj2,Rel,Score,Oldacp),
Acp=\=Oldacp.

dynamic_mapped(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Score,Acp) :-
dynamic_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Score,Acp).

dynamic_mapped(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Score,Acp) :-
rec_dynmapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Rel,Score,Oldacp),
Acp=\=Oldacp.

/* The following is a partly separate program which processes object
   mappings during several phases of analogous matching. These programs
   are used in several places using different components.

/* Rules which identify analogous mappings between specific pairs of
   objects. This rule is required in this window by the property-matching
   rule, since mapped objects must first be analogous before the mapping
   can be achieved. */

object_matches(Tobj,Sobj,Totalscore,Acp) :-
findall(Score,object_match(Tobj,Sobj,Score,Acp),Scorelist),
object_scores(Scorelist,0,Totalscore).

/* Sub-routines to identify different types of object match, based on
   different knowledge structures. These rules are mapped directly by
   the property matching program until a match is achieved. */

object_match(Tobj,Sobj,Score,Acp) :-
static_mapping(Tobj,_,Sobj,_,_,Score,Acp).

object_match(Tobj,Sobj,Score,Acp) :-
static_mapping(_,Tobj,_,Sobj,_,Score,Acp).

object_match(Tobj,Sobj,Score,Acp) :-
```

```
dynamic_mapping(Tobj,_,_,Sobj,_,_,_,Score,Acp).

object_match(Tobj,Sobj,Score,Acp) :-
dynamic_mapping(_,Tobj,_,_,Sobj,_,_,Score,Acp).

object_match(Tobj,Sobj,Score,Acp) :-
dynamic_mapping(_,_,Tobj,_,_,Sobj,_,Score,Acp).

/* Object-property matching program, called by the property_mapping
   routine, to check the validity of an object match using both matches
   between current ACP matches and previous matches */

object_matching(Tobj,Sobj,Acp) :-
object_match(Tobj,Sobj,_,Acp),!.

object_matching(Tobj,Sobj,Acp) :-
rec_objectmatch(Tobj,Sobj,_,_).

/* The following routine identifies mappings between labels on
    movements. This program is thesaurus-based, and uses membership
    of relevant lists to identify equivalent functionality on the movement
    of processes. A mapping occurs if structure-mapping has already
    identified an analogous mapping. A second-level rule is used to search
    all the possible lists of equivalent functionality. */

function_mapping(Tmvmt,Acp) :-
dynamic_mapping(Tobj1,Tobj2,Tobj3,Sobj1,Sobj2,Sobj3,Relation,_,Acp),
acp_ddata(Amvmt,Sobj1,Sobj2,Sobj3,Relation,Acp),
target_ddesc(Tmvmt,Tobj1,Tobj2,Tobj3,Relation),
not rec_funcmapping(Amvmt,_),
function_list(Samelist),
on(Amvmt,Samelist),
on(Tmvmt,Samelist).

/* Second-level lists, which represent a thesaurus describing the
   equivalence of state transitions in the models. */

function_list(Flist) :-
Flist=[loan,borrow,dispatch,send,lend,goods_out].

function_list(Flist) :-
Flist=[receipt,input,goods_in,arrival,addition].

function_list(Flist) :-
Flist=[allocate,assign,place,connect,join].

function_list(Flist) :-
Flist=[return,finish_loan].

function_list(Flist) :-
Flist=[record].

function_list(Flist) :-
Flist=[check_position,monitor].
```

```
/* Rules to determine interrelated structure of analogous match
   the rules are quite lenghy because they are Acp-dependent */

/* First layer of rules to determine the type of analogous
   similarity between two object-relation structures */

perfect_structure(Matched_acp) :-
calc_structure(Matched_acp,Matching_score),
Matching_score >= 81.

good_structure(Matched_acp) :-
calc_structure(Matched_acp,Matching_score),
Matching_score >=50,
Matching_score < 81.

poor_structure(Matched_acp) :-
calc_structure(Matched_acp,Matching_score),
Matching_score >=33,
Matching_score < 50.

fail_structure(Matched_acp) :-
calc_structure(Matched_acp,Matching_score),
Matching_score <33.

/* Second layer of rules to calculate the %age match for selected_acp. An
   initial rule is required to identify the need for a division by zero
   (equivalence to 100% perfect match). */

calc_structure(Selected_acp,100) :-
acp_total_mappings(Selected_acp,0),!.

calc_structure(Selected_acp,Relation_score) :-
relation_matches(Selected_acp,Total_score),
acp_total_mappings(Selected_acp,Total_mappings),
Relation_score is Total_score/Total_mappings * 100.

/* Third layers of rules to generate and count the relations list for the
   total of relations mappings */

relation_matches(Selected_acp,Relation_score) :-
findall(Relation,relation_match(Relation,Selected_acp),Relation_list),
length(Relation_list,Initial_score),
total_mapties(Selected_acp,Totalties),
Relation_score is Initial_score-Totalties.

/* Fourth layer of rules to determine extent of match with selected_acp */

relation_match(Relation,Selected_acp) :-
static_mapping(_,_,_,_,Relation,_,Selected_acp).

relation_match(Relation,Selected_acp) :-
dynamic_mapping(_,_,_,_,_,_,Relation,_,Selected_acp).

relation_match(Property,Selected_acp) :-
property_mapping(_,_,Property,Selected_acp).

relation_match(Condition,Selected_acp) :-
condition_mapping(_,_,_,_,_,_,_,_,_,_,_,_,_,Condition,Selected_acp).
```

```
relation_match(Movement,Selected_acp) :-
function_mapping(Movement,Selected_acp).

/* Fourth-level routine to identify mapping ties - it runs the same two
   routines for structure-mapping and dynamic-mapping to specify ties,
   which are then used to determine the score by which the total number
   of mappings must be reduced. This is the number of mappings which
   occur divided by two because each tie will be identified twice, once for
   each version. */

/* Total level rule counts the number of tie firings and divides by 2 to
    correct this total. */

total_mapties(Acp,Totalties) :-
findall(Acp,mapping_tie(Acp),Tlist),
length(Tlist,T),
Totalties is T/2.

/* Second-level rule identify both instances of tie-mappings which may
   occur. */

mapping_tie(Acp) :-
staticmapping_tie(Acp).

mapping_tie(Acp) :-
dynamicmapping_tie(Acp).

/* Third-level rules, which identify the specific instances of ties, which
    are copied from the static- and dynamic-mapping rules. */

staticmapping_tie(Acp) :-
acp_sdata(Sobj1,Sobj2,Srelation,Acp),
target_sdesc(Tobj1,Tobj2,Trelation),
compare(=,Srelation,Trelation),
not rec_statmapping(T1,T2,Sobj1,Sobj2,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
static_structure(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Score,Acp),
static_possibles(Sobj1,Sobj2,Tobj1,Tobj2,Trelation,Scores,Acp),
sort(Scores,Sorted_scores,[],1),
Sorted_scores = [Other_score|Rest],
Score = Other_score.

dynamicmapping_tie(Acp) :-
acp_ddata(_,Sobj1,Sobj2,Sobj3,Srelation,Acp),
target_ddesc(_,Tobj1,Tobj2,Tobj3,Trelation),
compare(=,Srelation,Trelation),
not rec_dynmapping(T1,T2,T3,Sobj1,Sobj2,Sobj3,Trelation,Any),
not detailed_object(Tobj1,Acp),
not detailed_object(Tobj2,Acp),
not detailed_object(Tobj3,Acp),
dynamic_structure(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Score,Acp),
dynamic_possibles(Sobj1,Sobj2,Sobj3,Tobj1,Tobj2,Tobj3,
Trelation,Scores,Acp),
sort(Scores,Sorted_scores,[],1),
Sorted_scores = [Other_score|Rest],
Score = Other_score.
```

```
/* Program to select the most appropriate level of the target structure
   for the first-pass matching. The aim of the program is to select the
   most appropriate level of the target domain so that it matches the high
   level descriptions of the abstract ACPs. The heuristic is to only select
   components at one level of detail below the space components. It
   attempts to identify components in the structural space rather than
   objects, so an additional qualifier is built into the rule, to ensure that
   any low-abstraction feature in a slot rather than object, i.e. the feature
   contains\has another feature. This program identifies abstraction levels
   of objects rather than structure, so host programs identify object levels
   within a structure. */

detailed_object(Low_object,Acp) :-
father(top,Acp),
target_sdata(High_object,Middle_object,R1),
target_sdata(Middle_object,Low_object,R2),
target_sdata(Low_object,Base_object,R3),
Rlist = [has_no,has_one,has_many,contains_no,contains_one,
contains_many],on(R1,Rlist),on(R2,Rlist),on(R3,Rlist).

/* Target restructuring program. It infers missing structure in the target
    problem from the existing description of the problem.
   This is useful for two functions: (i) identify existing target facts to
   assist the matching process, (ii) identify additional facts to assist in
   correct levelling of the problem during the initial matching process.
   It uses the structure of the target (A->B,B->C so A->C) to identify
   obligatory structural combinations from the known structure of the
   domain.

   The following programs work at several layers:
   i)  contains_many --> contains_many,
   ii) contains_one/has_many --> contains_many,
   iii) contains_one/has_one --> contains_one,
   iv)  has_one/many --> has_rules combinations. */

/* Top-level rule to identify two sources of the target description. */

target_sdesc(Tobj1,Tobj2,Relation) :-
target_sdata(Tobj1,Tobj2,Relation).

target_sdesc(Tobj1,Tobj2,Relation) :-
restructure_target(Tobj1,Tobj2,Relation,_),
Tobj1=\=world.

/* Series of second-level rules for possible combinations. The
   reconstructed static structure also includes the bypassed object, to
   permit these programs to be used in other routines described below. */

restructure_target(Tobj1,Tobj3,contains_many,Tobj2) :-
target_sdata(Tobj1,Tobj2,contains_many),
target_sdata(Tobj2,Tobj3,Any_relation),
not target_sdata(Tobj1,Tobj3,contains_many),!.

restructure_target(Tobj1,Tobj3,contains_many,Tobj2) :-
target_sdata(Tobj1,Tobj2,Any_relation),
target_sdata(Tobj2,Tobj3,contains_many),
not target_sdata(Tobj1,Tobj3,contains_many),!.

restructure_target(Tobj1,Tobj3,contains_many,Tobj2) :-
```

```
target_sdata(Tobj1,Tobj2,contains_one),
target_sdata(Tobj2,Tobj3,has_many),
not target_sdata(Tobj1,Tobj3,contains_many),!.

restructure_target(Tobj1,Tobj3,contains_many,Tobj2) :-
target_sdata(Tobj1,Tobj2,has_many),
target_sdata(Tobj2,Tobj3,contains_one),
not target_sdata(Tobj1,Tobj3,has_many),!.

restructure_target(Tobj1,Tobj3,has_many,Tobj2) :-
target_sdata(Tobj1,Tobj2,Any_relation),
target_sdata(Tobj2,Tobj3,has_many),
not target_sdata(Tobj1,Tobj3,has_many),!.

restructure_target(Tobj1,Tobj3,has_many,Tobj2) :-
target_sdata(Tobj1,Tobj2,has_many),
target_sdata(Tobj2,Tobj3,Any_relation),
not target_sdata(Tobj1,Tobj3,has_many),!.

restructure_target(Tobj1,Tobj3,has_one,Tobj2) :-
target_sdata(Tobj1,Tobj2,has_one),
target_sdata(Tobj2,Tobj3,has_one),
not target_sdata(Tobj1,Tobj3,has_one),!.

/* Similar rule to permit the restructuring of the requirements
   descriptions for the target domain. */

target_reqts(Tobj1,Tobj2,Relation) :-
target_reqt(Tobj1,Tobj2,Relation).

target_reqts(Tobj1,Tobj2,Relation,Value) :-
target_reqt(Tobj1,Tobj2,Relation,Value).

target_reqts(Tobj1,Tobj2,Relation) :-
target_reqt(Tobj1,Tobj3,Relation),
restructure_target(Tobj1,Tobj2,Relation,Tobj3).

target_reqts(Tobj1,Tobj2,Relation,Value) :-
target_reqt(Tobj1,Tobj3,Relation,Value),
restructure_target(Tobj1,Tobj2,Relation,Tobj3).

target_reqts(Tobj2,Tobj1,Relation,Value) :-
target_reqt(Tobj3,Tobj1,Relation,Value),
restructure_target(Tobj2,Tobj1,Relation,Tobj3).

/* A similar program is used to reconstruct dynamic knowledge from
   the structural changes which occur. The first rule identifies which
   can be replaced in the target domain. The rule is: replace(new,old). */

replace_objects(Tobj1,Tobj2) :-
restructure_target(Tobj1,Tobj3,Relation,Tobj2).

/* A top-level rule to redescribe the dynamic target domain, in a similar
   way that the static domain is described. There are several options for
   restructuring the dynamic movements, so they must be catered for
   in a series of rules. Due to nature of the OBJECT moved in dynamic
   structures, it cannot be restructured and so is not catered for. This
   leaves us with three possible options for the reconstructed dynamic
   target domain: replace 1st object, replace 2nd object, & replace both
```

```
1st & 2nd objects. */

target_ddesc(Function,Tobj1,Tobj2,Tobj3,Relation) :-
target_ddata(Function,Tobj1,Tobj2,Tobj3,Relation).

target_ddesc(Function,Tobj1,Tobj2,Tobj3,Relation) :-
target_ddata(Function,Tobj1,Tobj4,Tobj3,Relation),
replace_objects(Tobj2,Tobj4),
not target_ddata(Function,Tobj1,Tobj2,Tobj3,Relation).

target_ddesc(Function,Tobj1,Tobj2,Tobj3,Relation) :-
target_ddata(Function,Tobj1,Tobj2,Tobj4,Relation),
replace_objects(Tobj3,Tobj4),
not target_ddata(Function,Tobj1,Tobj2,Tobj3,Relation).

target_ddesc(Function,Tobj1,Tobj2,Tobj3,Relation) :-
target_ddata(Function,Tobj1,Tobj4,Tobj3,Relation),
replace_objects(Tobj2,Tobj4),
target_ddata(Function,Tobj1,Tobj2,Tobj5,Relation),
replace_objects(Tobj3,Tobj5),
not target_ddata(Function,Tobj1,Tobj2,Tobj3,Relation).
```

# Descriptions of the Problem Elicitation Dialogue within the Problem Identifier

```
/* The conditions elicitation program involves several levels of window,
   to allow to installation of a number of menus (3 in all), which can be
   used to offer a choice of static knowledge states, dynamic knowledge
   states and a value for the condition. */

/* Window definition */

conditions_window('Conditions Window') :-
wgcreate('Conditions Window',40,0,440,570,70,0,0,1,0),
setup_winF('Conditions Window'),
gviewer('Conditions Window',off),
wfront('Conditions Window').

setup_winF(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
present_conditions(Presconditions),
add_tools(Win,[
conditions(textbox('Chicago',12,0,4,0,32,32,1,'Enter Cond- ition')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_requirements(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinF,[
box(25,5,145,260),
box(175,5,245,260),
box(25,270,135,210),
box(165,270,210,210),
textline('Times',14,1,5,90,'Identifying Conditions on System Functions'),

textline('Times',12,1,30,30,'Conditions on System Functions'),
textbox('Times',12,0,45,10,36,250,0,'Some system functions only occur under specific conditions. This
window allows you to identify such conditions for appropriate functions.'),
textbox('Times',12,0,87,10,36,250,0,'To enter a condition double click ENTER CONDITION, then select a
function and the most appropriate condition for the function.'),
textbox('Times',12,0,129,10,24,250,0,'Ira limits you to only selecting conditions for two of your system
functions.'),

textline('Times',12,1,180,65,'Possible Conditions'),
textbox('Times',12,0,195,10,36,250,0,'Four types of condition are available for triggering system
functions:'),
textline('Times',12,0,225,10,'*'),
textline('Times',12,0,261,10,'*'),
textline('Times',12,0,297,10,'*'),
textline('Times',12,0,357,10,'*'),
textbox('Times',12,0,225,15,36,245,0,'MINIMUM_QTY: the function occurs when an associated entity
has reached a minimal level of contents,'),
textbox('Times',12,0,261,15,36,245,0,'MAXIMUM_QTY: the function occurs when an associated entity
has reached a maximum level of contents,'),
textbox('Times',12,0,297,15,60,245,0,'SAME_PROPERTIES: the function occurs when two associated
entities have the same values, for example a theatre-goer is allocated to a seat if it meets his needs, i.e. seat
price & requirements are both less than >£20,'),
textbox('Times',12,0,357,15,36,245,0,'DATE_LIMIT: the function occurs when a given date or time is
reached, or after a specific length of time.'),

textline('Times',12,1,30,360,'Hints'),
textline('Times',12,0,45,275,'*'),
textbox('Times',12,0,45,285,24,190,0,'Think to yourself why each function in the system occurs,'),
textline('Times',12,0,75,275,'*'),
textbox('Times',12,0,75,285,36,190,0,'A selected condition must apply to all instances of the function,'),
```

```
textline('Times',12,0,105,275,'*'),
textbox('Times',12,0,105,285,24,190,0,'Ira tentatively suggests the following condition:'),
textbox('Times',12,1,135,285,24,190,1,Presconditions),

textline('Times',12,1,170,320,'Personnel Example'),
box(200,280,80,185),
textline('Times',12,2,203,420,'World'),
speckled(fillcircle(240,305,20)),
speckled(fillcircle(240,425,22)),
fillbox(230,295,10,10),
fillbox(241,306,10,10),
fillbox(222,420,10,10),
fillbox(243,412,10,10),
fillbox(230,429,10,10),
fillbox(248,427,10,10),
textline('Times',12,2,260,395,'Organisation'),
textline('Times',12,2,260,282,'Agency'),
line((240,330),(240,350)),
line((235,345),(240,350)),
line((245,345),(240,350)),
fillbox(235,360,10,10),
line((240,375),(240,400)),
line((235,395),(240,400)),
line((245,395),(240,400)),
textline('Times',12,2,245,350,'Many'),
textline('Times',12,2,223,350,'Staff'),
textbox('Times',12,0,300,275,60,200,0,'None of the four available conditions control the arrival and
departure of staff from the organisation, so no conditions are allocated to the Record function.')]),
wkill('Categories Window'),
enable_item('Other Inputs','Add Condition'),
enable_item('Other Inputs','Del Condition').

/* Subroutine necessary to determine likely condition based on best
   function selection. */

present_conditions(P) :-
get_prop(acp,selection1,Acp),
acp_cdata(F,C,Acp),
concat(F,':',A),
concat(A,C,P),!.

present_conditions('Ira is uncertain of conditions !!') :- !.

/* This program describes the program to elicit all three types of
   knowledge from the user to construct the condition - note the use of
   set-prop to store and pass data between the 4 screens. */

conditions(double,Win) :-
del_prop(cond,fn),
del_prop(cond,val),
mdialog(48,78,220,240,
[button(190,10,20,140,'Create Condition'),
button(190,170,20,60,'Cancel'),
text(10,10,80,220,'Use both buttons to call a menu from which to select definitions of each condition. You
are advised to select the menus in the order given, then click CREATE CONDITION:'),
button(110,50,20,150,'Function'),
button(140,50,20,150,'Condition Value'),
],Button,condition_menu),
get_prop(cond,fn,F),
```

```
get_prop(cond,val,Value),
assertz(target_cdata(F,Value)).

/* Sub-windows containing windows for the three options in the main
   window. The first rules control the firing of the buttons, then validation
   control rules */

condition_menu(D,4) :- !,
condition_menu1(F),
set_prop(cond,fn,F),fail.
condition_menu(D,5) :- !,
condition_menu2(Value),
set_prop(cond,val,Value),fail.

condition_menu(D,B) :-
not get_prop(cond,fn,F),
beep(60),message(['You must select details using both buttons before trying to create the condition']),!,fail.
condition_menu(D,B) :-
not get_prop(cond,val,Value),
beep(60),message(['You must select details using both buttons before trying to create the condition']),!,fail.
condition_menu(D,B) :-
findall(Mvmt,target_cdata(Mvmt,_),Clist),
length(Clist,L),L>=2,
beep(60),message(['You have already created two conditions - delete existing conditions']),!,fail.
condition_menu(D,B) :-
get_prop(cond,fn,F),
get_prop(cond,val,Value),
target_cdata(F,Value),
beep(60),message(['This condition is already known to Ira']),!,fail.
condition_menu(D,1) :- !.

/* This first window offers a menu of the movements. To display
   movements on the list it is necessary to concat data into single data
   atoms: 1) concat each data, then 2) use findall to put this data in the
   list. get_ddata puts the target_ddata in the right format */

/* Two rules for eliciting data about the movement. The additional rule
    is necessary to identify when no target_ddata exist to construct a menu
    scroll bar for the necessary selection. Check rules are also included
    after each of these sub-dialogues. */

condition_menu1(F) :-
not target_ddata(_,_,_,_,_),beep(60),
mdialog(250,300,200,300,
[button(170,100,20,100,'Continue'),
text(10,10,96,280,'You have not yet input any functions from which to select. A condition on a function
cannot be created until it has been input to the system.')],Btn),!.

condition_menu1(F) :-
findall(Mvmt,target_ddata(Mvmt,_,_,_,_),Datalist),
Datalist = [First|Rest],
mdialog(250,300,200,300,
[button(170,30,20,60,'Ok'),
button(170,210,20,60,'Cancel'),
text(10,10,32,280,'Select the required function between two knowledge states:'),
menu(80,30,66,240,Datalist,[First],Mlist)],Btn,check_cmenu(Mlist)),
Mlist = [F|Allrest].

check_cmenu(D,B,Mlist) :-
```

```
length(Mlist,Total),Total=\=1,
beep(30),message(['You should select one function from the menu']),!,fail.

check_cmenu(D,B,Mlist) :-
Mlist=[Func|R],target_cdata(Func,_),
beep(30),message(['This function has already been given a condition']),!,fail.

check_cmenu(D,B,_) :- !.

/* Program to display and elicit the values for a selection */

condition_menu2(Value) :-
mdialog(250,300,200,200,
[button(170,20,20,60,'Ok'),
button(170,120,20,60,'Cancel'),
text(10,10,48,180,'Select the appropriate value of the condition:'),
menu(80,30,66,140,[minimum_qty,maximum_qty,same_properties,date_limit],[minimum_qty],Vlist)],Btn,
check_value(Vlist)),
Vlist = [Value|Rest].

check_value(D,B,Vlist) :-
length(Vlist,Total),Total =\= 1,
beep(30),message(['You should select one condition value from the menu']),!,fail.
check_value(D,B,_) :- !.

/* Routine to pass control to the next window */

pass_requirements(double,Win) :-
requirements_window('Requirements Window').
```

```
/* This is the final window of the knowledge elicitation process, which
   remains as a background to all remaining inputs\searching\
   examination by the analyst during the process. The window also has a
   function to check the completeness of solutions - see at the end of the
   window. */

/* Window definition. */

final_window('Searching\Update Window') :-
wgcreate('Searching\Update Window',40,0,440,570,0,0,0,1,0),
setup_winL('Searching\Update Window'),
gviewer('Searching\Update Window',off),
wfront('Searching\Update Window').

setup_winL(Win) :-
gsplit(Win,0),
gcursor(Win,hand),
control_menu,
add_tools(Win,[
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem'))],1),
add_pic(Win,picwinL,[
box(25,5,170,260),
box(200,5,220,260),
box(25,270,395,280),
textline('Times',14,1,5,110,'Searching & Updating your Problem Description'),

textline('Times',12,1,30,110,'Searching'),
textbox('Times',12,0,45,10,60,250,0,'Once the description of the domain has been input Ira can be
instructed to search for appropriate abstractions and reusable specifications. This is achieved by using the
SEARCH selection on the CONTROL menu (see below).'),
textbox('Times',12,0,117,10,36,250,0,'The description of your domain can also be altered at any time using
the two menus OBJECTS & OTHER INPUTS, as described below.'),

textline('Times',12,1,205,35,'Updating the Domain Description'),
textbox('Times',12,0,220,10,48,250,0,'The OBJECT & OTHER INPUTS menus can be used to change the
definition of the new application at any time. However, there are two instances during which this definition
cannot be changed:'),
textline('Times',12,0,274,10,'*'),
textbox('Times',12,0,274,20,24,240,0,'if the definition supports other facts about the domain, or'),
textline('Times',12,0,298,10,'*'),
textbox('Times',12,0,298,20,24,220,0,'it is a basis for analogous mappings resulting from a search (see
CONTROL menu).'),

textline('Times',12,1,30,340,'The CONTROL menu'),
textbox('Times',12,0,45,275,24,260,0,'The Control menu offers you several options regarding Iras
searching abilities:'),
textline('Times',12,2,73,275,'SEARCH'),
textbox('Times',12,0,85,285,36,260,0,'Search matches the current domain description to identify the most
likely abstraction for the domain. To do this Ira deletes any previous analogous matches.'),
textline('Times',12,2,127,275,'ABSTRACTION'),
textbox('Times',12,0,139,285,36,260,0,'Abstraction retrieves an explanation of the previously-matched
abstraction if Ira had successfully matched the domain.'),
textline('Times',12,2,181,275,'IDENTIFY MAPPINGS'),
textbox('Times',12,0,193,285,60,260,0,'Identify Mappings allows you to impose specific analogous
mappings with a mapped abstraction before rematching the domain description. This facility allows you to
experiment with the matching mechanism and overcome incorrect mappings identified by Ira.'),
textline('Times',12,2,259,275,'SEE TARGET'),
textbox('Times',12,0,271,285,12,260,0,'See a description of your domain.'),
textline('Times',12,2,288,275,'CONSISTENCY CHECKER'),
```

```
textbox('Times',12,0,300,285,24,260,0,'Ask Ira to check your problem description before attempting a
search.'),
textline('Times',12,2,330,275,'RESET SEARCH'),
textbox('Times',12,0,342,285,24,260,0,'Delete the mappings with the previously matched abstraction.'),
textline('Times',12,2,372,275,'NEW APPLICATION'),
textbox('Times',12,0,384,285,24,260,0,'Delete the description of the current problem domain.'),
]),wkill('Physical Window').

/* Two controls to help the user develop a more complete model by doing
   some basic consistency checks on the input model, with regard to the
   all-important structural featrures. */

consistency_check :-
unheaded_objects(O1,O2,O3,O4),
write_unheadobjects(O1,O2,O3,O4,A,B,C,D),
missing_objrelations(Olist),
write_missingobjects(Olist,E,F,G,H),
mdialog(48,78,380,350,
[button(350,125,20,100,'Continue'),
text(10,10,64,330,'Ira believes that you may want to address the following aspects of your description
before searching the knowledge base. Consider each fact carefully before changing your description:'),
text(90,10,32,330,'If necessary, use ADD STRUCTURE to input to following facts:'),
text(130,20,16,310,A),text(150,20,16,310,B),
text(170,20,16,310,C),text(190,20,16,310,D),
text(220,10,32,330,'Also use ADD STRUCTURE to possibly input to following structural relations between
objects:'),
text(260,20,16,310,E),text(280,20,16,310,F),
text(300,20,16,310,G),text(320,20,16,310,H)
],Btn).

/* Here is the routine to identify relational spaces between object
   structures and state transtion definitions. */

missing_objrelations(Olist) :-
findall((O1,O2),missing_relations(O1,O2),Olist).

missing_relations(O1,O2) :-
target_ddata(F,O1,O2,_,_),
not target_sdata(O2,O1,_).

missing_relations(O1,O2) :-
target_ddata(F,O1,_,O2,_),
not target_sdata(O2,O1,_).

/* Two subroutines to write objects on the dialogue screen. They are
   both sequential and simple. */

write_unheadobjects(O1,O2,O3,O4,A,B,C,D) :-
writeA(O1,A),writeB(O2,B),writeB(O3,C),writeB(O4,D).

writeA(O,A) :- O=\='',concat('world has_one/many ',O,A),!.
writeA(O,A) :- O='',A='No changes necessary'.

writeB(O,B) :- O=\='',concat('world has_one/many ',O,B),!.
writeB(O,B) :- B=''.

write_missingobjects(Olist,E,F,G,H) :-
Olist=[],E='No changes necessary',F='',G='',H='',!.
```

```
write_missingobjects(Olist,E4,F4,G4,H4) :-
length(Olist,1),Olist=[E],E=(E1,E2),
concat(E2,' contains_one/many ',E3),
concat(E3,E1,E4),F4='',G4='',H4='',!.

write_missingobjects(Olist,E4,F4,G4,H4) :-
length(Olist,2),Olist=[E,F],E=(E1,E2),F=(F1,F2),
concat(E2,' contains_one/many ',E3),
concat(E3,E1,E4),
concat(F2,' contains_one/many ',F3),
concat(F3,F1,F4),G4='',H4='',!.

write_missingobjects(Olist,E,F,G,H) :-
length(Olist,3),Olist=[E,F,G],
E=(E1,E2),F=(F1,F2),G=(G1,G2),
concat(E2,' contains_one/many ',E3),
concat(E3,E1,E4),
concat(F2,' contains_one/many ',F3),
concat(F3,F1,F4),
concat(G2,' contains_one/many ',G3),
concat(G3,G1,G4),H4='',!.

write_missingobjects(Olist,E,F,G,H) :-
length(Olist,4),Olist=[E,F,G,H],
E=(E1,E2),F=(F1,F2),G=(G1,G2),H=(H1,H2),
concat(E2,' contains_one/many ',E3),
concat(E3,E1,E4),
concat(F2,' contains_one/many ',F3),
concat(F3,F1,F4),
concat(G2,' contains_one/many ',G3),
concat(G3,G1,G4),
concat(H2,' contains_one/many ',H3),
concat(H3,H1,H4).
```

```
/* This program is quite complex, and attempts to elicit the dynamic
   structural relations between the objects. */

/* Window definition */

dynamic_window('Function Definition Window') :-
wgcreate('Function Definition Window',40,0,440,570,70,0,0,1,0),
setup_winD('Function Definition Window'),
gviewer('Function Definition Window',off),
wfront('Function Definition Window').

setup_winD(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
get_prop(function,list,L),
L=[Function|Rest],
prepare_selector(Function,Text),
prepare_introduction(Function,Intro),
add_tools(Win,[
dynamic_relations(textbox('Chicago',12,0,4,0,32,32,1,Text)),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
function_help(textbox('Chicago',12,0,6,0,32,32,1,'Function Help')),
stop_addfn(textbox('Chicago',12,0,4,0,32,32,1,'Restart Function Input')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem'))],1),
add_pic(Win,picwinD,[
box(25,5,140,260),
box(170,5,250,260),
box(25,270,155,210),
box(185,270,235,210),
textline('Times',14,1,5,170,Text),

textline('Times',12,1,30,70,'Function Definitions'),
textbox('Times',12,0,45,10,48,250,0,Intro),
textbox('Times',12,0,99,10,60,250,0,'To define a function double click DEFINE FUNCTION, then enter
the data requested by the dialogue. Before entering this data read the remainder of the guidelines given in this
window and sketch each function as suggested below.'),

textline('Times',12,1,175,60,'Defining Each Function'),
textbox('Times',12,0,190,10,36,250,0,'Each function is represented as a change in the state of an object.
Firstly sketch the function on paper using the following terms:'),
circle(252,70,25),
fillbox(242,50,10,10),
fillbox(257,70,10,10),
fillbox(240,75,10,10),
circle(252,200,25),
fillbox(242,180,10,10),
fillbox(257,200,10,10),
fillbox(237,205,10,10),
fillbox(252,135,10,10),
line((257,100),(257,130)),
line((257,150),(257,170)),
line((257,130),(252,125)),
line((257,130),(262,125)),
line((257,170),(252,165)),
line((257,170),(262,165)),
textline('Times',12,2,277,30,'Starting Position'),
textline('Times',12,2,277,165,'Final Position'),
textline('Times',12,2,227,115,'Processed'),
textline('Times',12,2,239,120,'Object'),
```

```
textline('Times',12,2,267,125,'Single'),

textbox('Times',12,0,293,10,48,250,0,'Each function processes one or many objects. When the function
occurs this object moves from one state (Starting Position) to a new state (Final Position).'),
textbox('Times',12,0,335,10,24,250,0,'You can view functions as physically moving the object from one
position to another in the domain.'),
textbox('Times',12,0,365,10,36,250,0,'For each function you should identify the processed object, its start
and final positions and the number of objects processed.'),
textline('Times',12,0,401,10,'See FUNCTION HELP for more guidance.'),

textline('Times',12,1,30,305,'Single or Many Objects'),
textbox('Times',12,0,45,275,60,190,0,'Each occurence of a function processes SINGLE or MANY
objects. Selecting a SINGLE object indicates that the function only processes one object at a time.'),
textbox('Times',12,0,105,275,36,190,0,'Selecting MANY objects represents the processing of more than
one object at any time.'),
textbox('Times',12,0,141,275,36,190,0,'You should identify whether each function processes SINGLE or
MANY objects each time the process is run.'),

textline('Times',12,1,190,320,'Personnel Example'),
speckled(fillcircle(240,315,20)),
speckled(fillcircle(240,435,20)),
fillbox(230,305,10,10),
fillbox(241,316,10,10),
fillbox(222,430,10,10),
fillbox(243,422,10,10),
fillbox(230,439,10,10),
fillbox(245,437,10,10),
textline('Times',12,2,260,405,'Organisation'),
textline('Times',12,2,260,292,'Agency'),
line((240,340),(240,360)),
line((235,355),(240,360)),
line((245,355),(240,360)),
fillbox(235,370,10,10),
line((240,385),(240,410)),
line((235,405),(240,410)),
line((245,405),(240,410)),
textline('Times',12,2,245,360,'Many'),
textline('Times',12,2,223,360,'Staff'),

textbox('Times',12,0,284,275,60,190,0,'The RECORD function records STAFF joining the organisation.
During RECORDING many staff move from the agency to the organisation, as represented diagrammatically
above.'),
textline('Times',12,0,350,275,'The functional definition is:'),
textline('Times',12,0,362,275,'* Object: Staff,'),
textline('Times',12,0,374,275,'* Start Position: Agency,'),
textline('Times',12,0,386,275,'* Final Position: Organisation,'),
textline('Times',12,0,398,275,'* Number: Many')]),
wkill('Function Examples Window'),
wkill('Structural Window').

/* Prepare_selector routine in order to describe the function narrative
   in pretty form for output on the left-hand selection boxes. Also routine
   to prepare the introduction paragraph. */

prepare_selector(Function,Text) :-
concat('Define ',Function,A),
concat(A,' Function',Text).

prepare_introduction(Function,Intro) :-
```

```
concat('Use this window to define the ',Function,A),
concat(A,' function selected in the previous window. This is achieved by describing the change which takes
place to an object when it is processed by that function.',Intro).

/* This program describes the program to elicit static structural relations
   to describe the new target problem. Note that this routine passes
   control directly the structure window without returning to the func.
   window. It does not remove the current function from the function list.
   This is carried out when structure for that function is being entered. */

dynamic_relations(double,Win) :-
get_prop(function,list,List),
List=[Func|Rest],
build_objects(Objlist),
remove(world,Objlist,Olist),
Olist = [O1,O2,O3,O4],
mdialog(48,78,280,400,
[button(250,130,20,60,'Create'),
button(250,230,20,60,'Cancel'),
text(10,10,80,380,'Consider the following function. You should identify the main object processed by the
function, its initial and final positions and the number of objects (Single vs Many) processed by the function.
Click CREATE to record this functional definition:'),
text(106,10,16,65,'Function:'),
text(106,75,16,150,Func),
text(140,10,16,120,'Processed Object:'),
edit(140,135,16,100,'',gread(Object)),
text(215,10,16,120,'Quantities Moved:'),
edit(215,135,16,40,'many',Rel),
text(165,80,16,55,'Initial:'),
edit(165,135,16,100,'',gread(Source)),
text(190,90,16,45,'Final:'),
edit(190,135,16,100,'',gread(Destination)),
text(106,260,16,110,'Known Entities:'),
text(136,275,16,100,O1),
text(152,275,16,100,O2),
text(168,275,16,100,O3),
text(184,275,16,100,O4),
],Button,dynamic_check(Rel,Object,Source,Destination)),
createall_objects(Object,Source,Destination),
translate_manyone(Rel,Relation),
assertz(target_ddata(Func,Object,Source,Destination,Relation)),
pass_static(double,Win).

/* Rules to constrain the maximum number of permitted dynamic
   relations. A sub-rule is required to calculate the total number of new
   entities which would have to be created when firing the rule (see below)
   */

dynamic_check(D,B,_,Object,Source,Destination) :-
findall(Obj,target_object(Obj),Tlist),
length(Tlist,Total1),
findall(Object,newfound_objects(Object,Source,Destination),Nlist),
length(Nlist,Total2),
Total is Total1+Total2,Total>5,
beep(60), message(['You are trying to identify more than 4 objects. Delete other objects first']),!,fail.

/* Rules to control input data, to validate and maintain consistency */

dynamic_check(D,B,_,Object,_,_) :-
```

```
Object='end_of_file',
beep(60), message(['You must enter the name of the object to be processed']),!,fail.

dynamic_check(D,B,_,_,Source,_) :-
Source='end_of_file',
beep(60), message(['You must enter the name of the starting position of the object']),!,fail.

dynamic_check(D,B,_,_,_,Destination) :-
Destination='end_of_file',
beep(60), message(['You must enter the name of the final position of the object']),!,fail.

dynamic_check(D,B,Rel,_,_,_) :-
Rel='end_of_file',
beep(60), message(['You must enter the type of movement (single or many']),!,fail.

dynamic_check(D,B,Rel,_,_,_) :-
Rel =\= 'single', Rel =\= 'many',
beep(60), message(['You must enter the number of objects moved (single or many)']),!,fail.

dynamic_check(D,B,_,Object,_,_) :-
not valid_character(Object),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),!,fail.

dynamic_check(D,B,_,_,Source,_) :-
not valid_character(Source),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),!,fail.

dynamic_check(D,B,_,_,_,Destination) :-
not valid_character(Destination),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),!,fail.

dynamic_check(D,B,Func,Rel,Object,Source,Destination) :-
target_ddata(Func,_,_,_,_),
beep(60),message(['This function has already been input into Ira. ~MPlease try again']),!,fail.

dynamic_check(D,B,_,_,_,_,_) :-
findall(Fn,target_ddata(Fn,_,_,_,_),Currents),
length(Currents,C),C>=4,
beep(60),message(['Ira is sorry but you have already input four functional definitions']),!,fail.

/* The following few rules ensure the consistency of the input
   describing the dynamic relation rules. Look for direction contradictions
   between rules in the same direction, then contradictions between
   opposing objects */

dynamic_check(D,B,Rel,Object,Source,Destination) :-
Rel='single',
target_ddata(_,Object,Source,Destination,'move_many'),
beep(60),message(['This number of moved objects contradicts previous functional definitions describing
your problem domain']),!,fail.

dynamic_check(D,B,Rel,Object,Source,Destination) :-
Rel='many',
target_ddata(_,Object,Source,Destination,'move_one'),
beep(60),message(['This number of moved objects contradicts previous functional definitions describing
your problem domain']),!,fail.

dynamic_check(D,B,_,_,_,_) :- !.
```

```
/* A subroutine is required for the creation of objects which may not
   already exist as domain objects. The program must ensure that all
   objects are processed when validating these components. */

createall_objects(Obj1,Obj2,Obj3) :-
findall(Obj1,createach_object(Obj1,Obj2,Obj3),Anylist).

createach_object(Obj,_,_) :-
not target_object(Obj),
assertz(target_object(Obj)).

createach_object(_,Obj,_) :-
not target_object(Obj),
assertz(target_object(Obj)).

createach_object(_,_,Obj) :-
not target_object(Obj),
assertz(target_object(Obj)).

/* A similar set of rules is required to calculate the number of uncreated
   entities to prempt the generation of more than four entities. */

newfound_objects(Obj1,Obj2,Obj3) :-
not target_object(Obj1).

newfound_objects(Obj1,Obj2,Obj3) :-
not target_object(Obj2).

newfound_objects(Obj1,Obj2,Obj3) :-
not target_object(Obj3).

/* A simple ruleset to translate input from the analyst to the move_one
   or move_many predicates required by the AE. */

translate_manyone(Rel,Relation) :-
Rel='single',Relation='move_one',!.

translate_manyone(Rel,Relation) :-
Rel='many',Relation='move_many',!.

/* The additional window required to provide examples of and support
   sketching of the system functions. It is a small window but has all the
   same characteristics as main windows, without tools. */

function_help(double,Win) :-
wgcreate('Function Help Window',40,0,440,355,70,0,0,1,0),
setup_winDH('Function Help Window'),
gviewer('Function Help Window',off),
wfront('Function Help Window').

setup_winDH(Win) :-
get_functionlist(F1,F2,F3,F4),
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
funchelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,picwinDH,[
box(25,5,115,260),
box(145,5,175,260),
```

```
box(325,5,95,260),
textline('Times',14,1,5,80,'Function Help'),

textline('Times',12,1,30,90,'Functions'),
textbox('Times',12,0,45,10,24,250,0,'You input the following functions:'),
textline('Times',14,0,63,90,F1),
textline('Times',14,0,81,90,F2),
textline('Times',14,0,97,90,F3),
textline('Times',14,0,115,90,F4),

textline('Times',12,1,150,80,'Example Sketch'),
textbox('Times',12,0,168,10,60,250,0,'You should sketch each system function before entering its
description into Ira. Below is the final sketch of the Record function for the personnel example. It suggests
the scope and scale of these sketches:'),

textline('Times',12,1,190,320,'Personnel Example'),
box(240,40,70,180),
textline('Times',12,2,228,190,'World'),
speckled(fillcircle(270,65,20)),
speckled(fillbox(250,165,40,40)),
fillbox(260,55,10,10),
fillbox(271,66,10,10),
fillbox(252,180,10,10),
fillbox(273,172,10,10),
fillbox(260,189,10,10),
fillbox(278,187,10,10),
textline('Times',12,2,290,155,'Organisation'),
textline('Times',12,2,290,42,'Agency'),
line((270,90),(270,110)),
line((265,105),(270,110)),
line((275,105),(270,110)),
fillbox(265,120,10,10),
line((270,135),(270,160)),
line((265,155),(270,160)),
line((275,155),(270,160)),
textline('Times',12,2,275,110,'Many'),
textline('Times',12,2,253,110,'Staff'),

textbox('Times',12,0,330,10,48,250,0,'Ira will display two windows for each function input. The first
window elicits a description of the function, then the second window requests further descriptions of certain
objects.'),
textbox('Times',12,0,386,10,24,250,0,'These two windows are displayed for each function input into
Ira.')]).

/* Routine to determine list of functions necessary to be displayed. */

get_functionlist(F1,F2,F3,F4) :-
get_prop(saved,list,List),
length(List,1),List=[F1],F2='',F3='',F4='',!.

get_functionlist(F1,F2,F3,F4) :-
get_prop(saved,list,List),
length(List,2),List=[F1,F2],F3='',F4='',!.

get_functionlist(F1,F2,F3,F4) :-
get_prop(saved,list,List),
length(List,3),List=[F1,F2,F3],F4='',!.

get_functionlist(F1,F2,F3,F4) :-
```

```
get_prop(saved,list,List),
length(List,4),List=[F1,F2,F3,F4],!.

/* Return tool for the function help window. */

funchelp_return(double,Win) :-
wkill('Function Help Window').

/* The program to pass control to the next window in the dialogue */

pass_static(double,Win) :-
structural_window('Structural Window').
```

```
/* This program elicits up to 4 functions from the analyst. Note that when
   the window is called the function list in the property (function,list,List)
   is created as an empty list - this is a vital feature of the window. */

/* Window definition */

functions_window('Functions Window') :-
wgcreate('Functions Window',40,0,440,570,70,0,0,1,0),
setup_winFF('Functions Window'),
gviewer('Functions Window',off),
wfront('Functions Window').

setup_winFF(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
add_function(textbox('Chicago',12,0,4,0,32,32,1,'Add Func- tion')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
stop_addfn(textbox('Chicago',12,0,4,0,32,32,1,'Restart Function Input')),
pass_dynamic(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinFF,[
box(25,5,127,260),
box(157,5,165,260),
box(170,270,175,210),
box(327,5,73,260),
textline('Times',14,1,5,135,'Identifying System Functions'),

textline('Times',12,1,30,70,'Selecting Functions'),
textbox('Times',12,0,45,10,60,250,0,'Select one or two functions which best represent the required
system. Focus on functions which support the major system goal identified in the previous window and
ignore functions which only occur in exceptional circumstances.'),
textbox('Times',12,0,111,10,36,250,0,'Select functions by double clicking on the command Add Function,
then choosing functions from the menu provided. Select one function at a time.'),

textline('Times',12,1,162,80,'Similar Functions'),
textbox('Times',12,0,177,10,48,250,0,'If you recognise two functions which are similar only select the
most appropriate function. Do not select several functions in Ira which are intended to represent one function
in your system, so:'),
textbox('Times',14,2,245,30,40,210,0,'Be Conservative When Selecting Functions !'),
textbox('Times',12,0,291,10,24,250,0,'Note that many systems may only have one major function.'),

textline('Times',12,1,330,80,'Restart Function Input'),
textbox('Times',12,0,345,10,48,250,0,'You may restart input of all functions by double-clicking
RESTART FUNCTION INPUT. Please note that all existing functions and structures are deleted when
restarting.'),

textline('Times',12,1,175,320,'A Simple Example'),
textbox('Times',12,0,190,275,48,200,0,'A simple example is provided to suggest how facts should be
entered into Ira. The example represents a typical personnel domain within an organisation.'),
textbox('Times',12,0,244,275,36,200,0,'The personnel system RECORDS staff who join the organisation,
so the major system function is:'),
textline('Times',12,0,284,275,'*    Record.'),
textbox('Times',12,0,302,275,36,200,0,'This function is selected from the list of functions provided when
Add Function command is double-clicked.'),

]),
res_open('iralogo'),
add_pic(Win,logoname,picture(40,310,100,145,resource(iralogo,iralogo))),
set_prop(function,list,[]),
```

```
wkill('Ira Introduction').

/* This program describes the program to elicit up to 4 functions of the
   target system. Functions are recorded in a list held in prop identified
   by (function,list,List), which is deleted and recreated with the new
   function every time. */

add_function(double,Win) :-
mdialog(48,78,200,300,
[button(170,30,20,60,'Save'),
button(170,210,20,60,'Cancel'),
text(10,10,48,280,'Please select one or two functions describing your system. Enter and SAVE one function
at a time:'),
menu(70,70,66,160,[loan,borrow,dispatch,send,lend,goods_out,receipt,input,goods_in,arrival,addition,all
ocate,assign,place,connect,join,return,finish_loan,monitor,check_position,record],[loan],Flist)],Btn,check_
functions(Flist)),
Flist=[Func|Rest],
get_prop(function,list,List),
del_prop(function,list),
Newlist=[Func|List],
set_prop(function,list,Newlist).

check_functions(D,B,Flist) :-
length(Flist,Length),Length=\=1,
beep(60),message(['Select one function at a time.~MPlease try again']),
!,fail.

check_functions(D,B,Flist) :-
get_prop(function,list,List),
length(List,Length),Length=2,
beep(60),message(['You have already selected two functions.~MPlease continue']),!,fail.

check_functions(D,B,Flist) :-
get_prop(function,list,List),
Flist=[Func],
on(Func,List),
beep(60),message(['You have already selected this function.~MPlease try again']),
!,fail.

check_functions(D,B,Flist) :-
Flist=[Func],target_ddata(Func,_,_,_,_),
beep(60),message(['You have already entered this function.~MPlease try again']),
!,fail.

check_functions(D,B,Flist) :- !.

/* Function to restart input of functions from the beginning, removing all
   from the earlier input. There is a two-part control to stop accidental
   deletions of inputs. It is accessed from four windows. */

stop_addfn(double,Win) :-
mdialog(100,150,110,300,
[text(10,10,64,280,'Note that restarting the input of function definitions with delete all existing definitions
!!'),
button(80,220,20,60,'Cancel'),
button(80,20,20,60,'Restart')],Btn),
retractall(target_ddata(_,_,_,_,_)),
retractall(target_sdata(_,_,_)),
remove_stopobjects,wkill(Win),
```

```
functions_window('Functions Window').

remove_stopobjects :-
retractall(target_object(_)),
assertz(target_object(world)).

/* Pass Control to the next window. Initially a check is made to ensure
   that at least one function has been entered, otherwise the remaining
   dialogue falls flat. When passing control store the list of functions in
   a second prop (saved,list,L), which is used for display purposes at a
   later date. */

pass_dynamic(double,Win) :-
get_prop(function,list,[]),
beep(60),mdialog(145,130,130,300,
[button(100,100,20,100,'Continue'),
text(20,20,64,260,'You must identify at least one system function before continuing to describe the
remainder of the domain.')],Btn),!.

pass_dynamic(double,Win) :-
get_prop(function,list,L),
reverse(L,R),
del_prop(function,list),
set_prop(function,list,R),
set_prop(saved,list,R),
funcexample_window('Function Examples Window').
```

```
/* This program elicits up to 4 functions from the analyst. Note that when
   the window is called the function list in the property (function,list,List)
   is created as an empty list - this is a vital feature of the window. */

/* Window definition */

funcexample_window('Function Examples Window') :-
wgcreate('Function Examples Window',40,0,440,570,70,0,0,1,0),
setup_winFE('Function Examples Window'),
gviewer('Function Examples Window',off),
wfront('Function Examples Window').

setup_winFE(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
printable_functions(Functions),
add_tools(Win,[
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
stop_addfn(textbox('Chicago',12,0,4,0,32,32,1,'Restart Function Input')),
pass_fromexample(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinFE,[
box(25,5,80,475),
box(110,5,130,475),
box(260,5,135,475),
textline('Times',14,1,5,135,'Similar Example Problems'),

textline('Times',12,0,30,10,'Your system functions were:'),
textline('Times',12,1,42,20,Functions),
textbox('Times',12,0,54,10,48,220,0,'Your problem should be similar to one of the two example problems
presented here. If not you may wish to re-enter your system functions.'),
textbox('Times',12,0,30,250,72,220,0,'Note the format of the two examples, You will be required to define
your problem in a similar format. Sketch these examples for future reference before double-clicking NEXT
WINDOW.')]),
get_prop(function,list,L),L=[Function|Rest],
mainfunctions(Function,Selection1,Selection2),
set_prop(acp,selection1,Selection1),
set_prop(acp,selection2,Selection2),
display_topexample(Selection1,Win),
display_bottomexample(Selection2,Win),
wkill('Functions Window').

/* A short, concat routine is included to get the input functions in a
   listable form. */

printable_functions(F) :-
get_prop(function,list,L),
length(L,1),L=[F],F2='',!.

printable_functions(F) :-
get_prop(function,list,L),
length(L,2),L=[F1,F2],
concat(F1,' & ',A),
concat(A,F2,B),
concat(B,'.',F).

/* Two sets of routines to display the four possible selections of the
   available - first set for the top selection, second set for the bottom
   selection. */
```

```
/* Top-level: OCP Display */

display_topexample(ocp,Win) :-
add_pic(Win,picwinDT1,[
textline('Times',12,1,120,210,'First Example'),
speckled(fillbox(150,30,60,60)),
speckled(fillcircle(180,200,30)),
fillbox(170,40,13,13),
fillbox(185,60,13,13),
fillbox(165,70,13,13),
fillbox(165,180,13,13),
fillbox(180,200,13,13),
line((180,95),(180,165)),
line((175,160),(180,165)),
line((185,160),(180,165)),
fillbox(165,120,13,13),
textline('Times',12,0,210,20,'Store of Items'),
textline('Times',12,0,210,160,'Source for Items'),
textbox('Times',12,0,150,270,72,200,0,'This example describes a simple system in which items and held
then leave a store to some outside source. The aim of the system is to maintain a store of objects which can be
used.')]),!.

/* Top-level: OMP Display */

display_topexample(omp,Win) :-
add_pic(Win,picwinDT2,[
textline('Times',12,1,120,210,'First Example'),
speckled(fillbox(150,30,60,100)),
speckled(fillbox(150,140,60,100)),
fillbox(170,40,20,20),
fillbox(170,75,20,20),
fillbox(170,180,20,20),
line((180,95),(180,165)),
line((175,160),(180,165)),
line((185,160),(180,165)),
textline('Times',12,0,135,40,'Trains'),
line((140,70),(165,80)),
textbox('Times',12,0,210,30,36,100,1,'Track section protecting trains against collisions'),
textbox('Times',12,0,210,140,36,100,1,'Track section protecting trains against collisions'),
textbox('Times',12,0,150,270,36,200,0,'This example describes a system which controls the safety of
trains moving along tracks in order to avoid collisions.'),
textbox('Times',12,0,186,270,36,200,0,'Only one train is permitted in each section, and the signalman is
warned whenever a track section contains two or more trains.')]),!.

/* Top-level: OPP Display */

display_topexample(opp,Win) :-
add_pic(Win,picwinDT3,[
textline('Times',12,1,120,210,'First Example'),
speckled(fillbox(150,30,60,100)),
speckled(fillbox(150,140,60,100)),
fillbox(170,75,20,20),
line((180,95),(180,165)),
line((175,160),(180,165)),
line((185,160),(180,165)),
textline('Times',12,0,135,40,'Boat'),
line((140,70),(165,80)),
textbox('Times',12,0,210,30,24,100,1,'Zone patrolled by coastguard boat'),
textbox('Times',12,0,210,140,24,100,1,'Zone patrolled by coastguard boat'),
```

```
textbox('Times',12,0,150,270,36,200,0,'This example describes a system which monitors the position of coastguard boats to ensure they maintain a tight cordon.'),
textbox('Times',12,0,186,270,36,200,0,'Each patrol zone is monitored to ensure it is being patrolled: if not the coordinator directs a boat to patrol that area.')]),!.

/* Top-level: OAP Display */

display_topexample(oap,Win) :-
add_pic(Win,picwinDT4,[
textline('Times',12,1,120,210,'First Example'),
speckled(fillbox(150,30,60,60)),
speckled(fillbox(150,170,60,60)),
hash(fillbox(170,40,13,13)),
hash(fillcircle(195,60,7)),
filloval(165,180,17,12),
fillbox(165,205,17,17),
hash(fillbox(167,207,13,13)),
fillcircle(195,210,11),
line((180,95),(180,165)),
line((175,160),(180,165)),
line((185,160),(180,165)),
hash(fillbox(165,120,13,13)),
textline('Times',12,0,210,20,'Theatre bookings'),
textbox('Times',12,0,210,130,24,140,1,'Seats for performance, containing bookings'),
textbox('Times',12,0,150,270,72,200,0,'A simple theatre reservation system allocates seat bookings for performances. Allocation is constrained by seat availability and price, smoking etc (as indicated by different booking shapes in the figure).')]),!.

/* Bottom-level: OCP Display */

display_bottomexample(ocp,Win) :-
add_pic(Win,picwinDB1,[
textline('Times',12,1,270,205,'Second Example'),
speckled(fillbox(300,30,60,60)),
speckled(fillcircle(330,200,30)),
fillbox(320,40,13,13),
fillbox(335,60,13,13),
fillbox(315,70,13,13),
fillbox(315,180,13,13),
fillbox(330,200,13,13),
line((330,95),(330,165)),
line((325,160),(330,165)),
line((335,160),(330,165)),
fillbox(315,120,13,13),
textline('Times',12,0,360,20,'Store of Items'),
textline('Times',12,0,360,160,'Source for Items'),
textbox('Times',12,0,300,270,72,200,0,'This example describes a simple system in which items and held then leave a store to some outside source. The aim of the system is to maintain a store of objects which can be used.')]),!.

/* Bottom-level: OMP Display */

display_bottomexample(omp,Win) :-
add_pic(Win,picwinDB2,[
textline('Times',12,1,270,205,'Second Example'),
speckled(fillbox(300,30,60,100)),
speckled(fillbox(300,140,60,100)),
fillbox(320,40,20,20),
fillbox(320,75,20,20),
```

```
fillbox(320,180,20,20),
line((330,95),(330,165)),
line((325,160),(330,165)),
line((335,160),(330,165)),
textline('Times',12,0,285,40,'Trains'),
line((290,70),(315,80)),
textbox('Times',12,0,360,30,36,100,1,'Track section protecting trains against collisions'),
textbox('Times',12,0,360,140,36,100,1,'Track section protecting trains against collisions'),
textbox('Times',12,0,300,270,36,200,0,'This example describes a system which controls the safety of
trains moving along tracks in order to avoid collisions.'),
textbox('Times',12,0,336,270,36,200,0,'Only one train is permitted in each section, and the signalman is
warned whenever a track section contains two or more trains.')]),!.

/* Bottom-level: OPP Display */

display_bottomexample(opp,Win) :-
add_pic(Win,picwinDB3,[
textline('Times',12,1,270,205,'Second Example'),
speckled(fillbox(300,30,60,100)),
speckled(fillbox(300,140,60,100)),
fillbox(320,75,20,20),
line((330,95),(330,165)),
line((325,160),(330,165)),
line((335,160),(330,165)),
textline('Times',12,0,285,40,'Boat'),
line((290,70),(315,80)),
textbox('Times',12,0,360,30,24,100,1,'Zone patrolled by coastguard boat'),
textbox('Times',12,0,360,140,24,100,1,'Zone patrolled by coastguard boat'),
textbox('Times',12,0,300,270,36,200,0,'This example describes a system which monitors the position of
coastguard boats to ensure they maintain a tight cordon.'),
textbox('Times',12,0,336,270,36,200,0,'Each patrol zone is monitored to ensure it is being patrolled: if not
the coordinator directs a boat to patrol that area.')]),!.

/* Bottom-level: OAP Display */

display_bottomexample(oap,Win) :-
add_pic(Win,picwinDB4,[
textline('Times',12,1,270,205,'Second Example'),
speckled(fillbox(300,30,60,60)),
speckled(fillbox(300,170,60,60)),
hash(fillbox(320,40,13,13)),
hash(fillcircle(345,60,7)),
filloval(315,180,15,10),
fillbox(320,205,13,13),
fillcircle(345,210,7),
line((330,95),(330,165)),
line((325,160),(330,165)),
line((335,160),(330,165)),
hash(fillbox(315,120,13,13)),
textline('Times',12,0,360,20,'Theatre bookings'),
textbox('Times',12,0,360,150,24,100,1,'Seats available for performance'),
textbox('Times',12,0,300,270,72,200,0,'A simple theatre reservation system allocates seat bookings for
performances. Allocation is constrained by seat availability and price, smoking etc (as indicated by different
booking shapes in the figure).')]),!.

/* Pass Control to the next window - simple and nothing complex here. */

pass_fromexample(double,Win) :-
dynamic_window('Function Definition Window').
```

```
/* This window is includes several programs to initialise the data input
   and to elicit some basic target problem information from the user.
   Initial interaction uses a window describing Ira, then calls several
   windows to accept data input */

/* Initialising window definition */

initial_window('Ira Introduction') :-
wgcreate('Ira Introduction',40,0,440,570,70,0,0,1,0),
setup_winA('Ira Introduction'),
gviewer('Ira Introduction',off),
wfront('Ira Introduction').

setup_winA(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
name_target(textbox('Chicago',12,0,4,0,32,32,1,'Enter System Name')),
elicit_goal(textbox('Chicago',12,0,4,0,32,32,1,'Enter System Goal')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_function(textbox('Chicago',12,0,6,0,32,32,1,'Next Window')),
pass_control(textbox('Chicago',12,0,6,0,32,32,1,'Go to Search'))],1),
add_pic(Win,picwinA,[
box(25,5,132,240),
box(162,5,258,240),
box(25,250,260,230),
box(290,250,130,230),
textline('Times',18,1,5,150,'Welcome to Ira'),

textline('Times',14,1,25,70,'What is Ira ?'),
textbox('Times',12,0,45,10,48,230,0,'Ira (the Intelligent Reuse Advisor) will help you to specify a new
computer system by retrieving old specifications for you to reuse. Ira has two major functions:'),
textline('Times',12,0,111,10,'ii) Searching for a reusable specification.'),
textline('Times',12,0,99,10,'i)  Eliciting facts about a new problem,'),
textbox('Times',12,0,129,10,24,230,0,'Initially Ira will guide you to input a description of the problem
which you wish to solve.'),

textline('Times',14,1,162,20,'How to Input Facts using Ira'),
textbox('Times',12,0,182,10,60,230,0,'Ira provides windows to input different facts about the problem,
Each window provides a description and examples of the facts to be entered. Two sets of commands exist,
one for fact entry and one for fact modification:'),
textbox('Times',12,0,248,20,24,220,0,'fact entry is achieved by double-clicking commands on the left of
the window,'),
textline('Times',12,0,248,10,'*'),
textline('Times',12,0,272,10,'*'),
textbox('Times',12,0,272,20,24,220,0,'modifying facts is achieved by pulling down two menus to the right
of the window.'),
textbox('Times',12,0,302,10,36,230,0,'Every window also has three additional commands found on this
and most other windows:'),
textline('Times',12,0,344,10,'*'),
textline('Times',12,0,368,10,'*'),
textline('Times',12,0,392,10,'*'),
textbox('Times',12,0,344,20,24,220,0,'General Help provides an overview of the facts which are input at
each window,'),
textbox('Times',12,0,368,20,36,220,0,'See Target Problem allows you to see facts input through earlier
windows,'),
textbox('Times',12,0,392,20,24,220,0,'Next window allows you to move onto the next window for data
input.'),
```

```
textline('Times',14,1,25,290,'The First Window'),
textbox('Times',12,0,45,255,72,220,0,'This first window requires you to input the name and most important goal of the new system. The system name is the name which your problem will be called by Ira. The main system goal provides a context with which to view future data input.'),
textbox('Times',12,0,123,255,60,220,0,'To input data double click on the command on the left of this window. A dialogue requesting the data to be input will appear. Clicking SAVE will record any data input, while CANCEL will abandon the input.'),

textbox('Times',12,0,189,255,60,220,0,'The system name and goal can be modified using the OTHER INPUTS menu - try this and see. Greyed-out menu options become available as you enter the relevant data about the problem.'),
textbox('Times',12,0,255,255,24,220,0,'To complete this window enter the system goal and name, then click Next Window.'),

textline('Times',14,1,290,270,'The Next Four Windows'),
textbox('Times',12,0,315,255,72,220,0,'The next 4 windows encourage you to model the main functions of the system. Initially you will be asked to sketch each function on paper, then Ira will ask you to input descriptions of these sketched functions into Ira, i.e.:'),
textbox('Times',14,2,383,295,35,160,1,'Sketch first, then input the description !')]),

enable_menu('Other Inputs'),
enable_item('Other Inputs','Mod Name'),
enable_item('Other Inputs','Mod Goal').


/* The program to elicit the name of the target system */

name_target(deactivate,Win) :-
gcursor(Win,hand).

name_target(double,Win) :-
mdialog(48,78,160,250,
[button(130,30,20,60,'Save'),
button(130,160,20,60,'Cancel'),
text(10,10,64,230,'Please enter the name of the new problem domain, then click SAVE:'),
edit(80,25,16,200,'',gread(Targetname))],Btn,check_name(Targetname)),
assertz(target_name(Targetname)).

check_name(D,B,Targetname) :-
target_name(X),nonvar(X),
beep(30),message(['The system has already been named']),!,fail.

check_name(D,B,Targetname) :-
not valid_character(Targetname),
beep(30),message(['The system name must begin with a small letter and only contain letters or numbers']),!,fail.

check_name(D,B,_) :- !.

/* The program to elicit the purpose or goal of the target system */

elicit_goal(deactivate,Win) :-
gcursor(Win,hand).

elicit_goal(double,Win) :-
mdialog(120,78,160,250,
[button(130,30,20,60,'Save'),
button(130,160,20,60,'Cancel'),
```

```
text(10,10,48,230,'Please input the most important goal of the new system, then click SAVE:'),
edit(80,25,16,200,'',gread(Targetgoal))],Btn,check_goal(Targetgoal)),
assertz(target_goal(Targetgoal)).

check_goal(D,B,Goal) :-
target_goal(X),nonvar(X),
beep(60),message(['The goal of the system already exists']),
!,fail.
check_goal(D,B,Goal) :-
not valid_character(Goal),
beep(60),message(['The system goal must begin with a small letter and only contain letters or
numbers']),!,fail.
check_goal(D,B,Goal) :- !.

/* The program to pass control to the next window in the dialogue, or to
   the search mechanism at the end of the data input phase (requiring all
   the menus to their selecttions to be enabled. There is the additional
   complication involving existence of already 4 functions, so functions
   loop must be avoided. */

pass_function(double,Win) :-
findall(F,target_ddata(F,_,_,_,_),Flist),length(Flist,4),
enable_item('Objects','Add Structure'),
enable_item('Objects','Del Structure'),
enable_item('Objects','Add Extra Object'),
enable_item('Objects','Delete Extra Object'),
wkill(Win),structures_window('Structures Window'),!.

pass_function(double,Win) :-
wkill(Win),functions_window('Functions Window').

pass_control(double,Win) :-
wkill(Win),
enable_menu('Other Inputs'),
enable_menu('Objects'),
enable_item('Other Inputs','Mod Name'),
enable_item('Other Inputs','Mod Goal'),
enable_item('Other Inputs','Add Condition'),
enable_item('Other Inputs','Del Condition'),
enable_item('Other Inputs','Add Label'),
enable_item('Other Inputs','Del Label'),
enable_item('Other Inputs','Add Reqt'),
enable_item('Other Inputs','Del Reqt'),
enable_item('Other Inputs','Add Scope'),
enable_item('Other Inputs','Del Scope'),
enable_item('Other Inputs','Add Physical'),
enable_item('Other Inputs','Del Physical'),
enable_item('Objects','Mod Object'),
enable_item('Objects','Change Categories'),
enable_item('Objects','Add Structure'),
enable_item('Objects','Del Structure'),
enable_item('Objects','Add Function'),
enable_item('Objects','Del Function'),
enable_item('Objects','Add Extra Object'),
enable_item('Objects','Delete Extra Object'),
control_menu,final_window('Searching\Update Window').
```

```
/* A simple window to elicit up to three labels from the analyst describing
   general features of the target system */

/* Initialising window definition */

label_window('Label Window') :-
wgcreate('Label Window',40,0,440,570,70,0,0,1,0),
setup_winJ('Label Window'),
gviewer('Label Window',off),
wfront('Label Window').

setup_winJ(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
elicit_labels(textbox('Chicago',12,0,6,0,32,32,1,'Enter Labels')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_physical(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinI,[
box(25,5,130,260),
box(175,270,245,210),
textline('Times',14,1,5,110,'Labels to Describe the Problem Domain'),

textline('Times',12,1,30,90,'What are Labels'),
textbox('Times',12,0,45,10,36,250,0,'Labels are general descriptors of information system problems. You
should select up to three labels which best describe the current problem domain.'),
textbox('Times',12,0,87,10,48,250,0,'To input a label describing your problem double click ENTER
LABELS then select the most relevant label(s) to describe the domain.'),

textline('Times',12,1,180,320,'Personnel Example'),
box(215,280,65,185),
textline('Times',12,2,203,420,'World'),
speckled(fillcircle(240,305,20)),
speckled(fillcircle(240,425,22)),
fillbox(230,295,10,10),
fillbox(241,306,10,10),
fillbox(222,420,10,10),
fillbox(243,412,10,10),
fillbox(230,429,10,10),
fillbox(248,427,10,10),
textline('Times',12,2,260,395,'Organisation'),
textline('Times',12,2,260,282,'Agency'),
line((240,330),(240,350)),
line((235,345),(240,350)),
line((245,345),(240,350)),
fillbox(235,360,10,10),
line((240,375),(240,400)),
line((235,395),(240,400)),
line((245,395),(240,400)),
textline('Times',12,2,245,350,'Many'),
textline('Times',12,2,223,350,'Staff'),

textbox('Times',12,0,284,275,24,200,0,'Two labels can be identified to describe the Personnel domain:'),
textline('Times',12,0,314,275,'*'),
textline('Times',12,0,326,275,'*'),
textline('Times',12,0,314,285,'Object_recording'),
textline('Times',12,0,326,285,'Recording'),
textbox('Times',12,0,344,275,36,200,0,'These two labels are selected from the list obtained by double
```

```
clicking ENTER LABELS.')]),
res_open('iralogo'),
add_pic(Win,logoname,picture(40,310,100,145,resource(iralogo,iralogo))),
wkill('Scope Window'),
enable_item('Other Inputs','Add Label'),
enable_item('Other Inputs','Del Label').

/* The program to elicit the name of the target system */

elicit_labels(double,Win) :-
mdialog(48,78,200,350,
[button(170,30,20,60,'Save'),
button(170,260,20,60,'Cancel'),
text(10,10,64,330,'Please select one or more labels from the menu to describe the new system. You may
select up to 3 labels, and record them by clicking SAVE:'),
menu(80,20,66,310,[stock_control,object_containment,resource_management,renewable_resource_mgmt,li
brary_system,nonrenewable_resource_mgmt,object_hiring,space_occupation,single_object_containment,obj
ect_monitoring,collision_detection,plan_adherence,object_allocation,constraint_satisfaction,requirement_mat
ching,object_recording,recording],[stock_control],Selection)],Btn,check_label(Selection)),
assert_labels(Selection).

/* The following program is required to update 1,2 or 3 new labels, as
   input by the analyst */

assert_labels(Selection) :-
Selection = [Sel1,Sel2,Sel3|Rest],
assertz(target_label(Sel1)),
assertz(target_label(Sel2)),
assertz(target_label(Sel3)),!.

assert_labels(Selection) :-
Selection = [Sel1,Sel2|Rest],
assertz(target_label(Sel1)),
assertz(target_label(Sel2)),!.

assert_labels(Selection) :-
Selection = [Sel1|Rest],
assertz(target_label(Sel1)).

/* Checks on the firing of this rule are quite complex, & include controls
   on the total number of labels selected and existing, as well as checks on
   the existence of these labels already. Checking here relies strongly on
   the order of the checks - verify the third item in the list first, then cut
   if it fails, so that the program does not require any additional checking
   */

check_label(D,B,Selection) :-
length(Selection,Total),Total = 0,
beep(60),message(['You must choose at least one label from the menu']),!,fail.
check_label(D,B,Selection) :-
length(Selection,Total),Total > 3,
beep(60),message(['You can only choose up to three labels from the menu']),!,fail.
check_label(D,B,Selection) :-
length(Selection,Length),
findall(Labels,target_label(Labels),Lablist),
length(Lablist,Total),
Total + Length > 3,
beep(60),message(['You can only chosen to create a total of more than three labels']),!,fail.
check_label(D,B,Selection) :-
```

```
Selection = [Sel1,Sel2,Sel3|Rest],
target_label(Sel3),
beep(60),message(['Your third selection menu has already been selected as a label']),!,fail.
check_label(D,B,Selection) :-
Selection = [Sel1,Sel2|Rest],
target_label(Sel2),
beep(60),message(['Your second selection menu has already been selected as a label']),!,fail.
check_label(D,B,Selection) :-
Selection = [Sel1|Rest],
target_label(Sel1),
beep(60),message(['Your first selection menu has already been selected as a label']),!,fail.
check_label(D,B,Selection) :- !.

/* Control of access to the next window */

pass_physical(double,Win) :-
physical_window('Physical Window').
```

```
/* Programs to control modification menus and all calls from the General
   Menu */

/* This program is quite obvious - it creates the series of menus
   which permit the analyst to modify appropriate information. The
   menus are disabled initially, so that they can be introduced during
   the input dialogue */

create_menus :-
install_menu('Objects',['Mod Object;Add Function;Del Function;Add Structure;Del Structure;Change
Categories;Add Extra Object;Delete Extra Object']),
install_menu('Other Inputs',['Mod Name;Mod Goal;Add Condition;Del Condition;Add Reqt;Del Reqt;Add
Scope;Del Scope;Add Label;Del Label;Add Physical;Del Physical']),
disable_menu('Other Inputs'),
disable_menu('Objects'),
disable_item('Other Inputs','Mod Name'),
disable_item('Other Inputs','Mod Goal'),
disable_item('Other Inputs','Add Condition'),
disable_item('Other Inputs','Del Condition'),
disable_item('Other Inputs','Add Label'),
disable_item('Other Inputs','Del Label'),
disable_item('Other Inputs','Add Reqt'),
disable_item('Other Inputs','Del Reqt'),
disable_item('Other Inputs','Add Scope'),
disable_item('Other Inputs','Del Scope'),
disable_item('Other Inputs','Add Physical'),
disable_item('Other Inputs','Del Physical'),
disable_item('Objects','Mod Object'),
disable_item('Objects','Change Categories'),
disable_item('Objects','Add Structure'),
disable_item('Objects','Add Extra Object'),
disable_item('Objects','Delete Extra Object'),
disable_item('Objects','Del Structure'),
disable_item('Objects','Add Function'),
disable_item('Objects','Del Function').

/* Menu calls to programs which permit input of data from the menus */

'Other Inputs'('Mod Name') :- modify_name.
'Other Inputs'('Mod Goal') :- modify_goal.
'Other Inputs'('Add Condition') :- conditions(double,A).
'Other Inputs'('Del Condition') :- delete_condition.
'Other Inputs'('Add Label') :- elicit_labels(double,A).
'Other Inputs'('Del Label') :- delete_label.
'Other Inputs'('Add Reqt') :- requirements(double,A).
'Other Inputs'('Del Reqt') :- del_reqt.
'Other Inputs'('Add Scope') :- scope(double,A).
'Other Inputs'('Del Scope') :- del_scope.
'Other Inputs'('Add Physical') :- elicit_physical(double,A).
'Other Inputs'('Del Physical') :- del_physical.
'Objects'('Mod Object') :- modify_object.
'Objects'('Change Categories') :- properties(double,A).
'Objects'('Add Structure') :- static_relations(double,A).
'Objects'('Del Structure') :- del_structure.
'Objects'('Add Function') :- add_movement.
'Objects'('Del Function') :- del_movement.
'Objects'('Add Extra Object') :- add_object(double,A).
'Objects'('Delete Extra Object') :- delete_object.
```

```
/***** Modify the name of the system. There are two versions for this
       rule, the first for the case where no problem name entered,
       so the dialogue acts as a name creation screen. *****/

modify_name :-
not target_name(Oname),
centred(T,L,280,250),
mdialog(T,L,130,250,
[button(100,30,20,60,'Create'),
button(100,160,20,60,'Cancel'),
text(10,10,32,230,'Please enter the new name of the system, then click CREATE:'),
edit(60,10,16,230,'',gread(New_name))],Btn,
check_modname(New_name)),
assertz(target_name(New_name)),!.

modify_name :-
target_name(Old_name),
centred(T,L,280,250),
mdialog(T,L,130,250,
[button(100,30,20,60,'Modify'),
button(100,160,20,60,'Cancel'),
text(10,10,32,230,'Please enter the new name of the system, then click MODIFY:'),
edit(60,10,16,230,write(Old_name),gread(New_name))],Btn,
check_modname(New_name)),
assertz(target_name(New_name)),
retract(target_name(Old_name)).

check_modname(D,B,New_name) :-
New_name = 'end_of_file',
beep(60),message(['You must enter a new name']),!,fail.
check_modname(D,B,New_name) :- !.

/***** Modify the goal of the system. There is an optional window
       provided to stop goal modification if no goal had already been
       entered. *****/

modify_goal :-
not target_goal(Old_goal),
centred(T,L,280,250),
mdialog(T,L,110,250,
[button(80,75,20,100,'Continue'),
text(10,20,64,220,'You have not yet entered the system goal, so it cannot be modified.')],Btn),!.

modify_goal :-
target_goal(Old_goal),
centred(T,L,280,250),
mdialog(T,L,130,250,
[button(100,30,20,60,'Modify'),
button(100,160,20,60,'Cancel'),
text(10,10,32,230,'Please enter the changed goal of the system, then click MODIFY:'),
edit(60,10,16,230,write(Old_goal),gread(New_goal))],Btn,
check_modgoal(New_goal)),
assertz(target_goal(New_goal)),
retract(target_goal(Old_goal)),!.

check_modgoal(D,B,New_goal) :-
New_goal = 'end_of_file',
beep(60),message(['You must enter a new goal']),!,fail.
check_modgoal(D,B,New_goal) :- !.
```

```
/***** Select and delete a label for the system. An option is required
       in case of no labels currently existing to be deleted. *****/

delete_label :-
not target_label(L),
mdialog(160,140,90,250,
[button(60,75,20,100,'Continue'),
text(10,10,32,230,'There are currently no labels to be deleted.')],Btn).

delete_label :-
findall(L,target_label(L),List),
List=[First|Rest],
mdialog(160,140,170,260,
[button(140,30,20,60,'Delete'),
button(140,170,20,60,'Cancel'),
text(10,10,32,240,'Please select the label to be deleted then click DELETE:'),
menu(60,10,50,240,List,[First],Slist)
],Btn,check_delabel(Slist)),
Slist=[Label],retract(target_label(Label)).

check_delabel(D,B,Slist) :-
length(Slist,L),L=\=1,
beep(60),message(['You should select one label to delete']),!,fail.

check_delabel(D,B,_) :- !.

/*****
  Delete an existing definition of the problem scope. This program is a
  variation on a theme - i.e. construction and matching must use the scope
  list rather than target_ddata rules, so we require alternative subroutines
  to process these features. Delete scope also has an optional dialogue
  which identifies situations in which there are no scopes to delete.

*****/

del_scope :-
not target_scope(Fn),
mdialog(130,150,110,300,
[button(80,100,20,100,'Continue'),
text(10,10,48,280,'There are currently no functions which have been identified as beyond the scope of the
information system.')],Btn),!.

del_scope :-
findall(Data,target_scope(Data),Datalist),
Datalist = [First|Rest],
mdialog(58,125,210,260,
[button(180,20,20,100,'Delete Scope'),
button(180,180,20,60,'Cancel'),
text(10,10,48,240,'Select an object movement beyond the control of the computer system to be deleted, then
click DELETE:'),
menu(70,10,98,240,Datalist,[First],Mlist)],Btn,check_scope(Mlist)),
Mlist = [M],retract(target_scope(M)).

check_scope(D,B,Mlist) :-
length(Mlist,Total),Total =\= 1,
beep(60),message(['You must select one function from the menu']),!,fail.
check_scope(D,B,_) :- !.
```

```
/* Specialised program to match selected list contents to the original
   target item */

get_scope(T7) :-
target_scope(O1,O2,O3,R),
concat(',',R,T1),
concat(O3,T1,T2),
concat(',',T2,T4),
concat(O2,T4,T5),
concat(',',T5,T6),
concat(O1,T6,T7).

find_scope(O1,O2,O3,R,Selected) :-
target_scope(O1,O2,O3,R),
concat(',',R,T1),
concat(O3,T1,T2),
concat(',',T2,T4),
concat(O2,T4,T5),
concat(',',T5,T6),
concat(O1,T6,T7),
compare(=,T7,Selected).

/*****
   Deletion of an existing requirement. This is simplified from the
   addition program since there is no need for a second button, since
   values are shown on the initial window
*****/

del_reqt :-
not target_reqt(A,B,C),
not target_reqt(D,E,F,G),
mdialog(100,150,110,300,
[button(70,100,20,100,'Continue'),
text(10,10,36,280,'There are currently no requirements for the system to be deleted')],Btn),!.

del_reqt :-
findall(Data,get_reqt(Data),Datalist),
Datalist = [First|Rest],
mdialog(58,125,230,400,
[button(200,20,20,160,'Remove Requirement'),
button(200,320,20,60,'Cancel'),
text(10,10,64,380,'Select the requirement which you wish to undo, then click REMOVE
REQUIREMENT:'),
menu(80,50,98,300,Datalist,[First],List)],Btn,
check_delreqts(List)),
List = [L],find_reqt(Object1,Object2,Relation,Value,L),
retract_reqts(Object1,Object2,Relation,Value).

retract_reqts(Object1,Object2,Relation,Value) :-
on(Value,Vlist),
Vlist=[minimum_qty,maximum_qty,same_properties,date_limit],
retract(target_reqt(Object1,Object2,Relation,Value)),!.
retract_reqts(Object1,Object2,Relation,Value) :-
retract(target_reqt(Object1,Object2,Relation)).

check_delreqts(D,B,List) :-
length(List,Total),Total =\= 1,
beep(60),message(['You must select one requirement from menu']),!,fail.
check_delreqts(D,B,_) :- !.
```

```
/* Specialised version of the string-matching menu eliciter, to read the
   correct selection from the menu. This program is made more complex
   by the possibility of two types of requirement - those with and without
   values to the requirements */

/* Two rules to get both kinds of requirement into the menu */

get_reqt(T7) :-
target_reqt(O1,O2,R),
concat(',',R,T1),
concat(O2,T1,T5),
concat(',',T5,T6),
concat(O1,T6,T7).

get_reqt(T7) :-
target_reqt(O1,O2,R,V),
concat(',',V,T1),
concat(R,T1,T2),
concat(',',T2,T4),
concat(O2,T4,T5),
concat(',',T5,T6),
concat(O1,T6,T7).

/* Two programs to retranslate the selected menu item from the single
   atom to the original requirement */

find_reqt(O1,O2,R,V,Selected) :-
target_reqt(O1,O2,R,V),
concat(',',V,T1),
concat(R,T1,T2),
concat(',',T2,T4),
concat(O2,T4,T5),
concat(',',T5,T6),
concat(O1,T6,T7),
compare(=,T7,Selected),!.

find_reqt(O1,O2,R,V,Selected) :-
target_reqt(O1,O2,R),
concat(',',R,T1),
concat(O2,T1,T5),
concat(',',T5,T6),
concat(O1,T6,T7),
compare(=,T7,Selected).

/*******
   Deletion of a physical attribute from a simpler menu format than
   that required during the larger data items which needed to be
   selected. The usual optional dialogue exists to block dialogue if no
   physical attributes are available to deletion.
*****/

del_physical :-
not target_phyprop(A,B),
mdialog(110,150,110,300,
[button(70,100,20,100,'Continue'),
text(10,10,36,280,'There are currently no physical attributes to be deleted')],Btn),!.

del_physical :-
```

```
delcheck(D,3,F,C) :-
retract(target_cdata(F,C)),!.

delcheck(D,B,_,_) :- !.

/* Fetch conditions, which retrieve relevant condition for display. A
   set-prop counter is used to alternate between the condition which
   is displayed to the analyst at a time. The counter is originally set in
   the main program when Ira is accessed, i.e. in the bootup routine. */

fetch_condition(F,C) :-
findall((F,C),target_cdata(F,C),Clist),
length(Clist,1),Clist=[(F,C)],!.

fetch_condition(F,C) :-
findall((F,C),target_cdata(F,C),Clist),
length(Clist,2),fetch_condchoice(Clist,F,C),!.

fetch_condchoice(Clist,F,C) :-
get_prop(delete,condition,1),
set_prop(delete,condition,2),
Clist=[(F,C)|Rest],!.

fetch_condchoice(Clist,F,C) :-
get_prop(delete,condition,2),
set_prop(delete,condition,1),
reverse(Clist,Nlist),
Nlist=[(F,C)|Rest],!.
```

```
/* Programs called by the Objects Menu */

/* Add a new object to the data base */

add_object(double,A) :-
build_objects(Objlist),
Objlist=[O1,O2,O3,O4,O5],
mdialog(100,150,230,250,
[button(200,30,20,60,'Add'),
button(200,160,20,60,'Cancel'),
text(10,10,32,230,'Please enter the name of the new object, then click ADD:'),
edit(55,70,16,100,'',gread(Object)),
text(80,40,16,200,'Existing objects include:'),
text(100,70,16,100,O1),
text(116,70,16,100,O2),
text(132,70,16,100,O3),
text(148,70,16,100,O4),
text(164,70,16,100,O5)],Btn,
check_addobject(Object)),
assertz(target_object(Object)).

check_addobject(D,B,Object) :-
Object = 'end_of_file',
beep(60),message(['You must enter a new object']),!,fail.

check_addobject(D,B,Object) :-
target_object(Object),
beep(60),message(['I am sorry but this object already exists']),!,fail.

check_addobject(D,B,Object) :-
not valid_character(Object),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),!,fail.

check_addobject(D,B,Object) :-
findall(Objects,target_object(Objects),Objlist),
length(Objlist,Total),Total = 5,
beep(60),message(['You have already created 4 objects.~MPlease modify one of the existing objects']),!,fail.
check_addobject(D,B,Object) :- !.

/* Modify an existing object. This is quite a complex program because it
   requires the tool to make considerable modifications to the following
   knowledge structures:
   - object,
   - structure (x2),
   - function (x3),
   - requirement ((x2)x2),
   - property,
   - physical attribute. All these knowledge types must be changed a new
   object name is entered via modifications. */

modify_object :-
mdialog(120,140,180,250,
[button(150,30,20,60,'Modify'),
button(150,160,20,60,'Cancel'),
text(10,10,48,230,'Please enter the object to be changed and the new version of the object, then click
MODIFY:'),
text(70,20,16,100,'Old object:'),
edit(70,125,16,100,'',gread(Old_object)),
text(100,20,16,100,'New object:'),
```

```
edit(100,125,16,100,'',gread(New_object))],Btn,
check_modobject(Old_object,New_object)),
change_objects(Old_object,New_object).

check_modobject(D,B,Old_object,_) :-
Old_object = 'end_of_file',
beep(60),message(['You should enter objects for both fields']),!,fail.

check_modobject(D,B,_,New_object) :-
New_object = 'end_of_file',
beep(60),message(['You should enter objects for both fields']),!,fail.

check_modobject(D,B,Old_object,_) :-
not target_object(Old_object),
beep(60),message(['The old object is not recognised by the ~Msystem']),!,fail.

check_modobject(D,B,Old_object,_) :-
Old_object=world,
beep(60),message(['I am sorry but you cannot modify the WORLD object']),!,fail.

check_modobject(D,B,_,New_object) :-
not valid_character(New_object),
beep(60),message(['The new object must begin with a small letter and only contain letters or
numbers']),!,fail.

check_modobject(D,B,_,New_object) :-
target_object(New_object),
beep(60),message(['The new object already exists - try again']),!,fail.

check_modobject(D,B,_,New_object) :-
New_object=world,
beep(60),message(['You cannot enter the WORLD object - it exists already']),!,fail.

check_modobject(D,B,_,_) :- !.

/* The following suite of routines modify all the relevant knowledge
   structures linked to the name of the object. */

change_objects(O,N) :-
findall(O,change_object(O,N),Olist).

change_object(O,N) :-
retract(target_object(O)),
assertz(target_object(N)).

change_object(O,N) :-
retract(target_sdata(O,A,B)),
assertz(target_sdata(N,A,B)).

change_object(O,N) :-
retract(target_sdata(A,O,B)),
assertz(target_sdata(A,N,B)).

change_object(O,N) :-
retract(target_ddata(A,O,B,C,D)),
assertz(target_ddata(A,N,B,C,D)).

change_object(O,N) :-
retract(target_ddata(A,B,O,C,D)),
```

```
assertz(target_ddata(A,B,N,C,D)).

change_object(O,N) :-
retract(target_ddata(A,B,C,O,D)),
assertz(target_ddata(A,B,C,N,D)).

change_object(O,N) :-
retract(target_reqt(O,A,B)),
assertz(target_reqt(N,A,B)).

change_object(O,N) :-
retract(target_reqt(A,O,B)),
assertz(target_reqt(A,N,B)).

change_object(O,N) :-
retract(target_reqt(O,A,B,C)),
assertz(target_reqt(N,A,B,C)).

change_object(O,N) :-
retract(target_reqt(A,O,B,C)),
assertz(target_reqt(A,N,B,C)).

change_object(O,N) :-
retract(target_pdata(O,A)),
assertz(target_pdata(N,A)).

change_object(O,N) :-
retract(target_phyprop(O,A)),
assertz(target_phyprop(N,A)).

/* Delete an existing object. This routine only allows deletion of existing
   objects which were created as such and are no longer part of a
   functional definition, so the menu only displays such objects. Two
   sections of routine are provided, the first if no objects are suitable
   for deletion. */

delete_object :-
not get_extraobjects(Obj),
centred(T,L,280,250),
mdialog(T,L,110,250,
[button(80,75,20,100,'Continue'),
text(10,10,48,230,'There are currently no extra objects to be deleted - click CONTINUE')],Btn),!.

delete_object :-
findall(Obj,get_extraobjects(Obj),Olist),
Olist=[First|Rest],
centred(T,L,280,250),
mdialog(T,L,150,250,
[button(120,30,20,60,'Ok'),
button(120,160,20,60,'Cancel'),
text(10,10,32,230,'Please enter the name of the object to be deleted:'),
menu(50,50,50,150,Olist,[First],Slist)],Btn,
check_delobject(Slist)),Slist=[Object],
retract(target_object(Object)).

check_delobject(D,B,Slist) :-
length(Slist,L),L=\=1,
beep(60),message(['Please select one object from the list']),!,fail.
```

```
check_delobject(D,B,Slist) :-
Slist=[Object],used_object(Object),
beep(60),message(['This object cannot currently be deleted because it supports existing structure, mvmt,
properties & physical structure']),!,fail.

check_delobject(D,B,Slist) :- !.

/* An additional routine is required to determine all of the additional
   objects to put them in a list for the menu. */

get_extraobjects(Object) :-
target_object(Object),
Object=\=world,
not function_object(Object).

function_object(Object) :-
target_ddata(_,Object,_,_,_),!.
function_object(Object) :-
target_ddata(_,_,Object,_,_),!.
function_object(Object) :-
target_ddata(_,_,_,Object,_).

/* Properties Management Window - for all four objects simultaneously. It
   constructs the right screen using build objects & set properties to
   obtain the required property ons\offs */

manage_properties(double,A) :-
build_objects(Objlist),
Objlist = [Ob1,Ob2,Ob3,Ob4,Ob5],
findall(Result,set_properties(Objlist,Result),Proplist),
Proplist = [I5,I4,I3,I2,I1],
mdialog(48,78,260,250,
[button(230,30,20,60,'Ok'),
button(230,160,20,60,'Cancel'),
text(10,10,80,230,'Please click any object to change its status - ON implies that the object has properties
while OFF implies that object properties are not critical:'),
check(100,80,20,130,Ob1,I1,O1),
check(120,80,20,130,Ob2,I2,O2),
check(140,80,20,130,Ob3,I3,O3),
check(160,80,20,130,Ob4,I4,O4),
check(180,80,20,130,Ob5,I5,O5),
],Button),
set_prop(prop,i1,I1),
set_prop(prop,i2,I2),
set_prop(prop,i3,I3),
set_prop(prop,i4,I4),
set_prop(prop,i5,I5),
set_prop(prop,o1,O1),
set_prop(prop,o2,O2),
set_prop(prop,o3,O3),
set_prop(prop,o4,O4),
set_prop(prop,o5,O5),
set_prop(prop,ob1,Ob1),
set_prop(prop,ob2,Ob2),
set_prop(prop,ob3,Ob3),
set_prop(prop,ob4,Ob4),
set_prop(prop,ob5,Ob5),
findall(Obj,change_properties(Obj),Dlist).
```

```
/* Set_properties program is initially required to set the properties
   from the existing target_pdata facts. It uses findall to check each object
   in the objlist and put 'on' or 'off' in the resulting list, which is then read
   by the program to put the relevant check boxes on */

set_properties(Objlist,Result) :-
Objlist = [_,_,_,_,Obj],
nonvar(Obj),
target_pdata(Obj,_),
Result = 'on'.
set_properties(Objlist,Result) :-
Objlist = [_,_,_,_,Obj],
nonvar(Obj),
not target_pdata(Obj,_),
Result = 'off'.
set_properties(Objlist,Result) :-
Objlist = [_,_,_,Obj,_],
nonvar(Obj),
target_pdata(Obj,_),
Result = 'on'.
set_properties(Objlist,Result) :-
Objlist = [_,_,_,Obj,_],
nonvar(Obj),
not target_pdata(Obj,_),
Result = 'off'.
set_properties(Objlist,Result) :-
Objlist = [_,_,Obj,_,_],
nonvar(Obj),
target_pdata(Obj,_),
Result = 'on'.
set_properties(Objlist,Result) :-
Objlist = [_,_,Obj,_,_],
nonvar(Obj),
not target_pdata(Obj,_),
Result = 'off'.
set_properties(Objlist,Result) :-
Objlist = [_,Obj,_,_,_],
nonvar(Obj),
target_pdata(Obj,_),
Result = 'on'.
set_properties(Objlist,Result) :-
Objlist = [_,Obj,_,_,_],
nonvar(Obj),
not target_pdata(Obj,_),
Result = 'off'.
set_properties(Objlist,Result) :-
Objlist = [Obj,_,_,_,_],
nonvar(Obj),
target_pdata(Obj,_),
Result = 'on'.
set_properties(Objlist,Result) :-
Objlist = [Obj,_,_,_,_],
nonvar(Obj),
not target_pdata(Obj,_),
Result = 'off'.

/* Program called by findall to modify each of the property values if it
   has been altered during the use of the window. The cases for retract
   come first, then the cases for asserting a non-existent property */
```

```
/* Retraction programs */

change_properties(Ob1) :-
get_prop(prop,i1,I1),
get_prop(prop,o1,O1),
get_prop(prop,ob1,Ob1),
I1 \= O1,
O1 = 'off',
retract(target_pdata(Ob1,_)).

change_properties(Ob2) :-
get_prop(prop,i2,I2),
get_prop(prop,o2,O2),
get_prop(prop,ob2,Ob2),
I2 \= O2,
O2 = 'off',
retract(target_pdata(Ob2,_)).

change_properties(Ob3) :-
get_prop(prop,i3,I3),
get_prop(prop,o3,O3),
get_prop(prop,ob3,Ob3),
I3 \= O3,
O3 = 'off',
retract(target_pdata(Ob3,_)).

change_properties(Ob4) :-
get_prop(prop,i4,I4),
get_prop(prop,o4,O4),
get_prop(prop,ob4,Ob4),
I4 \= O4,
O4 = 'off',
retract(target_pdata(Ob4,_)).

change_properties(Ob5) :-
get_prop(prop,i5,I5),
get_prop(prop,o5,O5),
get_prop(prop,ob5,Ob5),
I5 \= O5,
O5 = 'off',
retract(target_pdata(Ob5,_)).

/* Assertion programs */

change_properties(Ob1) :-
get_prop(prop,i1,I1),
get_prop(prop,o1,O1),
get_prop(prop,ob1,Ob1),
I1 \= O1,
O1 = 'on',
assertz(target_pdata(Ob1,different_properties)).

change_properties(Ob2) :-
get_prop(prop,i2,I2),
get_prop(prop,o2,O2),
get_prop(prop,ob2,Ob2),
I2 \= O2,
O2 = 'on',
```

```
assertz(target_pdata(Ob2,different_properties)).

change_properties(Ob3) :-
get_prop(prop,i3,I3),
get_prop(prop,o3,O3),
get_prop(prop,ob3,Ob3),
I3 \= O3,
O3 = 'on',
assertz(target_pdata(Ob3,different_properties)).

change_properties(Ob4) :-
get_prop(prop,i4,I4),
get_prop(prop,o4,O4),
get_prop(prop,ob4,Ob4),
I4 \= O4,
O4 = 'on',
assertz(target_pdata(Ob4,different_properties)).

change_properties(Ob5) :-
get_prop(prop,i5,I5),
get_prop(prop,o5,O5),
get_prop(prop,ob5,Ob5),
I5 \= O5,
O5 = 'on',
assertz(target_pdata(Ob5,different_properties)).
```

```
/* This window describes the definition of initial 'Run Ira' menu, and
   the 'Control' menu for interaction with analysts once guided data
   input has been completed. This window also includes simple routines
   called by these menus. */

/* Initial Ira Menu, with one choice in order to get the system working,
   and another two choices to reset search and target predicates in the
   data base. An interrupt dialogue is included to avoid accidental
   deletions. */

'<LOAD>'(D) :-
install_menu('Run Ira',['Run Ira/R;Reset Search/S;New Application/N']).

'Run Ira'('Run Ira') :-
set_counters,
create_menus,
kill_menu('Run Ira'),
initial_window('Ira Introduction').

'Run Ira'('Reset Search') :-
removeall_dialogue1.

'Run Ira'('New Application') :-
removeall_dialogue2.

/* Control menu permitting searching, additional data input and
   browsing of the selected or partial matches determined by the
   analogy engine. A check routine is included to disable the 'Identify
   Mappings' routine if no current acpmatches exist. */

control_menu :-
install_menu('Control',['Search;Abstraction;Identify Mappings;See Target;General Help;Consistency
Checker;Reset Search;New Application;Quit from Ira']),
check_idmappings.

check_idmappings :-
not rec_acpmatch(Acp),
disable_item('Control','Identify Mappings'),!.
check_idmappings :- !.

'Control'('Search') :-
removeall_dialogue3,
searching_acps(top).

'Control'('Abstraction') :-
get_abstraction.

'Control'('Reset Search') :-
removeall_dialogue1.

'Control'('New Application') :-
removeall_dialogue2.

'Control'('Identify Mappings') :-
inputmapping_dialogue.

'Control'('See Target') :-
set_prop(menu,control,on),
menu_target.
```

```
'Control'('General Help') :-
general_help(double,A).

'Control'('Consistency Checker') :-
consistency_check.

'Control'('Quit from Ira') :-
quit_from_ira.

/* Subroutine to quit and reset Ira. */

quit_from_ira :-
wkill('Searching\Update Window'),
wkill('General Help Window'),
kill_menu('Objects'),
kill_menu('Other Inputs'),
kill_menu('Control'),
install_menu('Run Ira',['Run Ira/R;Reset Search/S;New Application/N']).

/* Deletion control dialogues - simply checking that analysts do not
   accidentally delete work\mappings. They insert an additional dialogue
   to give users a chance to quit before deleting important data. */

removeall_dialogue1 :-
mdialog(100,150,110,300,
[text(10,10,64,280,'Are you sure that you want to remove all previously identified analogous mappings ?'),
button(80,220,20,60,'Cancel'),
button(80,20,20,60,'Delete')],
Btn,removeall_check1).

removeall_check1(D,3) :-
removeall_mappings,!.

removeall_dialogue2 :-
mdialog(100,150,110,300,
[text(10,10,64,280,'Are you sure that you want to remove the existing target domain description ?'),
button(80,220,20,60,'Cancel'),
button(80,20,20,60,'Delete')],
Btn,removeall_check2).

removeall_check2(D,3) :-
banner(findall(_,removeall_target,_),['Please be patient - Ira is removing the definition of the current
problem'],150,110),!.

removeall_dialogue3 :-
mdialog(100,150,130,300,
[text(10,10,64,280,'Searching will delete all previous analogous mappings and begin matching from scratch
- Okay ?'),
button(100,200,20,80,'Halt'),
button(100,20,20,80,'Proceed')],
Btn,removeall_check3).

removeall_check3(D,3) :-
removeall_mappings,!.

/* The following routine is the dialogue to tell the analyst when no
   abstraction is available to analyse or retrieve the abstraction which
   was identified by the search mechanism. */
```

```
get_abstraction :-
rec_acpmatch(Selected_acp),
not father(Selected_acp,No_son),
fetch_explanation(Selected_acp),!.

get_abstraction :-
noabstraction_dialogue.

noabstraction_dialogue :-
mdialog(100,150,130,300,
[text(10,10,64,280,'Ira has not yet identified any retrieved abstractions - please use the search mechanism to
identify abstractions.'),
button(100,75,20,100,'Continue')],Btn).
```

```
/* This program is quite simple - it attempts to elicit up to 4 basic objects
   from the analyst, to guide the modelling of the target problem. */

/* Window definition */

objects_window('Objects Window') :-
wgcreate('Objects Window',40,0,440,570,70,0,0,1,0),
setup_winB('Objects Window'),
gviewer('Objects Window',off),
wfront('Objects Window').

setup_winB(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
add_object(textbox('Chicago',12,0,6,0,32,32,1,'Add Objects')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_static(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinB,[
box(25,5,170,260),
box(200,5,220,260),
box(210,270,210,210),
textline('Times',14,1,5,25,'Identifying Problem Objects'),

textline('Times',12,1,30,70,'Inputting Objects'),
textbox('Times',12,0,45,10,48,250,0,'In this window Ira requests you to identify up to four important
entities or objects in the problem domain. In particular you should identify objects which are linked to
meeting the major system goal.'),
textbox('Times',12,0,93,10,60,250,0,'You may be assisted in identifying important problem objects by
developing a simple entity- relationship model for the domain. Again, focus on the central objects of the
domain rather than those which are peripheral to meeting the system goal.'),
textbox('Times',12,0,153,10,36,250,0,'Enter objects by double clicking on the command Add Objects.
Modify and Delete Objects using the Objects menu.'),

textline('Times',12,1,205,70,'The WORLD object'),
textbox('Times',12,0,220,10,48,250,0,'Ira automatically creates a fifth entity with which to describe the
problem domain. The World entity represents the entire problem space, and is needed to describe the number
of other entities in the problem.'),
textbox('Times',12,0,268,10,60,250,0,'When identifying your four entities you are advised to ensure that
you do not create an entity which overlaps with Iras World entity.'),

textline('Times',12,1,215,320,'A Simple Example'),
textbox('Times',12,0,230,275,48,200,0,'A simple example is provided to suggest how facts should be
entered into Ira. The example represents a typical personnel domain within an organisation.'),
textbox('Times',12,0,284,275,72,200,0,'The major goal of the personnel system is to record data about
staff who work for then leave an organisation. Staff joins from agencies then leave the company. Some
important entities in modeling the personnel domain are:'),
textline('Times',12,0,360,275,'*  Staff,'),
textline('Times',12,0,372,275,'*  Organisation,'),
textline('Times',12,0,384,275,'*  Agency,'),
textline('Times',12,0,396,275,'*  Outside World.'),

speckled(fillbox(10,295,130,160)),
blank(fillbox(140,295,60,160)),
textbox('Bookman',14,2,147,296,51,158,1,'HINT: Draw an Entity-Relationship Diagram'),
blank(fillbox(40,365,20,30,5,5)),
blank(fillbox(70,395,20,30,5,5)),
blank(fillbox(100,365,20,30,5,5)),
```

```
line((60,385),(70,410)),
line((90,410),(100,380))]),

wkill('Ira Introduction').

/* This program describes the data input program to elicit an initial
   series of 3/4 critical objects describing the key aspects of the target
   system */

elicit_objects(double,Win) :-
mdialog(48,78,170,300,
[button(140,30,20,60,'Save'),
button(140,210,20,60,'Cancel'),
text(10,10,48,280,'Please input up to four objects or entities describing the system. Enter and SAVE one
object at a time:'),
text(80,40,16,60,'Object:'),
edit(80,110,16,100,'',gread(Object))],Btn,check_objects(Object)),
assertz(target_object(Object)).

check_objects(D,B,Object) :-
Object = 'end_of_file',
beep(60),message(['You must enter the name of the object']),
!,fail.

check_objects(D,B,Object) :-
target_object(Object),
beep(60),message(['You have already input this object']),
!,fail.

check_objects(D,B,Object) :-
not valid_character(Object),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),
!,fail.

check_objects(D,B,Object) :-
findall(Objects,target_object(Objects),Objlist),
length(Objlist,Length),
Length >= 5,
beep(60),message(['You have already input four items.~MPlease delete or modify existing objects']),
!,fail.

check_objects(D,B,Object) :- !.
```

```
/* A window to elicit physical attributes of specific objects in the problem
   */

/* Initialising window definition */

physical_window('Physical Window') :-
wgcreate('Physical Window',40,0,440,570,70,0,0,1,0),
setup_winK('Physical Window'),
gviewer('Physical Window',off),
wfront('Physical Window').

setup_winK(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
elicit_physical(textbox('Chicago',12,0,4,0,32,32,1,'Enter Physical Desc.')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_final(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinK,[
box(25,5,170,260),
box(200,5,220,260),
box(25,270,145,210),
box(175,270,245,210),
textline('Times',14,1,5,50,'Identifying Physical Attributes of the Domain'),

textline('Times',12,1,30,40,'Physical Attributes of the Domain'),
textbox('Times',12,0,45,10,72,250,0,'This is the final window to elicit facts about your application.
Matching your problem description can be enhanced by knowledge of physical domain features. You should
identify physical attributes of objects identified in earlier windows.'),
textbox('Times',12,0,117,10,48,250,0,'To enter a physical attribute double click ENTER PHYSICAL
DESC. then enter an object and select the most appropriate physical attribute for that object.'),

textline('Times',12,1,205,75,'Physical Attributes'),
textbox('Times',12,0,220,10,60,250,0,'Physical attributes represent known types of common applications.
These applications include Warehousing, Libraries and Air Traffic Control systems. Think of your specific
application when selecting physical attributes.'),
textbox('Times',12,0,288,10,36,250,0,'You can enter a maximum of five physical attributes, and only one
physical attribute for each object.'),

textline('Times',12,1,30,355,'Hints'),
textline('Times',12,0,45,275,'*'),
textbox('Times',12,0,45,285,24,190,0,'Think of the specific application in more detail.'),
textline('Times',12,0,75,275,'*'),
textbox('Times',12,0,75,285,36,190,0,'Sketch the physical features of the application to help you identify
attributes more clearly.'),
textline('Times',12,0,117,275,'*'),
textbox('Times',12,0,117,285,36,190,0,'Ensure that the attribute correctly represents the application before
entering it.'),

textline('Times',12,1,180,320,'Personnel Example'),
box(215,280,65,185),
textline('Times',12,2,203,420,'World'),
speckled(fillcircle(240,305,20)),
speckled(fillcircle(240,425,22)),
fillbox(230,295,10,10),
fillbox(241,306,10,10),
fillbox(222,420,10,10),
fillbox(243,412,10,10),
```

```
fillbox(230,429,10,10),
fillbox(248,427,10,10),
textline('Times',12,2,260,395,'Organisation'),
textline('Times',12,2,260,282,'Agency'),
line((240,330),(240,350)),
line((235,345),(240,350)),
line((245,345),(240,350)),
fillbox(235,360,10,10),
line((240,375),(240,400)),
line((235,395),(240,400)),
line((245,395),(240,400)),
textline('Times',12,2,245,350,'Many'),
textline('Times',12,2,223,350,'Staff'),

textbox('Times',12,0,284,275,72,200,0,'Attributes appropriate to the Personnel system are:'),
textline('Times',12,0,314,275,'* Organisation - company, or'),
textline('Times',12,0,326,275,'* Organisation - place_of_work.')]),
wkill('Label Window'),
enable_item('Other Inputs','Add Physical'),
enable_item('Other Inputs','Del Physical').

/* The program to elicit the name of the target system */

elicit_physical(double,Win) :-
build_objects(Objlist),
remove(world,Objlist,Olist),
Olist=[O1,O2,O3,O4],
mdialog(48,78,230,350,
[button(170,160,20,140,'Save Attribute'),
button(200,200,20,60,'Cancel'),
text(10,10,48,330,'Please select one object and a physical attribute for that object, then record the attribute by clicking SAVE ATTRIBUTE:'),
text(74,20,16,100,'Object:'),
edit(94,20,16,100,'',gread(Object)),
text(74,200,16,120,'Attributes'),
menu(90,140,66,200,[in_sequence,in_building,different_properties,are_borrowed,taken_away,is_container
,moves_physically,are_manned_vehicle,adjacent_in_space,construct_network,company,place_of_work],[in
_sequence],Selection),
text(119,20,16,100,'Objects are:'),
text(134,20,16,100,O1),
text(150,20,16,100,O2),
text(166,20,16,100,O3),
text(182,20,16,100,O4),
],Btn,check_physical(Object,Selection)),
Selection = [Sel|Rest],
assertz(target_phyprop(Object,Sel)).

check_physical(D,B,Object,Selection) :-
Object = 'end_of_file',
beep(60),message(['You must enter an object to which to add an attribute. ~MPlease try again']),!,fail.

check_physical(D,B,Object,Selection) :-
not target_object(Object),
beep(60),message(['This object is not known to the system. ~MPlease try again']),!,fail.

check_physical(D,B,Object,Selection) :-
length(Selection,Total),Total =\= 1,
beep(60),message(['You must choose one attribute from the menu']),!,fail.
```

```
check_physical(D,B,Object,Selection) :-
findall(Obj,target_phyprop(Obj,_),Objlist),
length(Objlist,L),L>=5,
beep(60),message(['You have already entered 5 physical attributes - delete or modify existing
attributes']),!,fail.

check_physical(D,B,Object,_) :-
target_phyprop(Object,_),
beep(60),message(['This object has already been allocated an attribute. ~MRemove the existing attribute
first']),!,fail.

check_physical(D,B,Object,Selection) :-
Selection = [Sel|Rest],
target_phyprop(Object,Sel),
beep(60),message(['This attribute has already been allocated to the object']),!,fail.

check_physical(D,B,Object,Selection) :- !.

/* Control of access to the next window */

pass_final(double,Win) :-
final_window('Searching\Update Window').
```

```
/* This program is simple in comparison to other programs, and
   permits the analyst to identify objects which are critically effected
   by properties. */

/* Window definition */

properties_window('Categories Window') :-
wgcreate('Categories Window',40,0,440,570,70,0,0,1,0),
setup_winE('Categories Window'),
gviewer('Categories Window',off),
wfront('Categories Window').

setup_winE(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
possible_categories(Presobject),
add_tools(Win,[
properties(textbox('Chicago',12,0,4,0,32,32,1,'Enter
Categ- ories')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_conditions(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinE,[
box(25,5,170,260),
box(200,5,220,260),
box(25,270,135,210),
box(165,270,255,210),
textline('Times',14,1,5,100,'Identifying Object Categories'),

textline('Times',12,1,30,70,'Object Properties'),
textbox('Times',12,0,45,10,60,250,0,'So far little has been said about the nature of objects identified in
earlier windows. This window suggests some features of these objects by categorising them. Select entities
which describe the roles of objects in the problem domain.'),
textbox('Times',12,0,115,10,36,250,0,'To enter an object category double click ENTER CATEGORIES
then enter the object and select the relevant category from the scroll menu.'),
textbox('Times',12,0,157,10,36,250,0,'You may only identify one category per object, and only identify
two objects which are categorised.'),

textline('Times',12,1,205,45,'Permitted Object Categories'),
textbox('Times',12,0,220,10,24,250,0,'The following object categories can be selected:'),
textline('Times',12,0,244,10,'*'),
textline('Times',12,0,304,10,'*'),
textline('Times',12,0,328,10,'*'),
textline('Times',12,0,364,10,'*'),
textbox('Times',12,0,244,15,60,245,0,'DIFFERENT_OBJECT_TYPES: each object may have many
different values and these values play an important role in processing the object, for example in a cinema
seating domain both the reservation and the seats must be of the same type,'),
textbox('Times',12,0,304,15,24,245,0,'RESOURCE_CONTAINER: the object is a container in which
other objects are held,'),
textbox('Times',12,0,328,15,36,245,0,'RESOURCE: the object acts as a resource with which system
requirements are fulfilled. Resources are often contained in a resource_container,'),
textbox('Times',12,0,364,15,36,245,0,'RECEPTICABLE: the object receives other objects, ie. it is their
final destination.'),

textline('Times',12,1,30,360,'Hints'),
textline('Times',12,0,45,275,'*'),
textbox('Times',12,0,45,285,24,190,0,'Only apply a category to an object if it applies to the object in all
cases,'),
textline('Times',12,0,75,275,'*'),
```

```
textbox('Times',12,0,75,285,24,190,0,'Only identify categories for objects dependent upon functions
identified earlier,'),
textline('Times',12,0,105,275,'*'),
textbox('Times',12,0,105,285,24,190,0,'Ira tentatively proposes the following object categorisation:'),
textline('Times',12,1,135,285,Presobject),

textline('Times',12,1,170,300,'Personnel Example'),
box(185,290,65,160),
speckled(fillcircle(210,315,20)),
speckled(fillcircle(210,390,22)),
fillbox(200,305,10,10),
fillbox(211,316,10,10),
fillbox(192,380,10,10),
fillbox(203,372,10,10),
fillbox(198,394,10,10),
fillbox(213,390,10,10),
textline('Times',12,2,230,355,'Organisation'),
textline('Times',12,2,230,292,'Agency'),
textline('Times',12,2,185,413,'World'),

textbox('Times',12,0,255,275,36,200,0,'There are no object categories which are applicable to the
personnel domain, for several reasons:'),
textline('Times',12,0,297,275,'*'),
textbox('Times',12,0,297,285,60,190,0,'DIFFERENT_OBJECT_TYPES: the type of staff (e.g. clerical or
mgmt) does not affect their joining the organisation,'),
textline('Times',12,0,345,275,'*'),
textbox('Times',12,0,345,285,24,190,0,'RESOURCE: Staff are not resources which populate the
organisation,'),
textline('Times',12,0,369,275,'*'),
textbox('Times',12,0,369,285,36,190,0,'RESOURCE_CONTAINER: the organisation does not treat staff
as a resource.')]),
wkill('Structures Window'),
enable_item('Objects','Change Categories').

/* A routine to determine possible problem categories based solely on the
   best match with the domain functionality. It uses the target_ddata
   match to guess object matches. To simplify the model just select one,
   'randomly-chosen' category. A dummy rule is included to ensure firing
   of the rule. */

possible_categories(Presobject) :-
get_prop(acp,selection1,Acp),
findall((O,C),acp_pdata(Acp,O,C),Plist),
Plist=[(Obj,Cat)|Rest],
fetch_tobject(Obj,Tobj),
concat(Tobj,':',A),
concat(A,Cat,Presobject),!.

possible_categories('No categories applicable').

fetch_tobject(Obj,Tobj) :-
acp_ddata(F,Obj,_,_,_,Acp),
target_ddata(F,Tobj,_,_,_),!.

fetch_tobject(Obj,Tobj) :-
acp_ddata(F,_,Obj,_,_,Acp),
target_ddata(F,_,Tobj,_,_),!.

fetch_tobject(Obj,Tobj) :-
```

```
acp_ddata(F,_,_,Obj,_,Acp),
target_ddata(F,_,_,Tobj,_).

/* This program describes the program to elicit static structural relations
   to describe the new target problem. It also permits the deletion and
   modification of properties as necessary. */

properties(double,Win) :-
build_objects(Objlist),
remove(world,Objlist,Olist),
Olist=[O1,O2,O3,O4],
mdialog(48,78,240,350,
[button(185,200,20,60,'Save'),
button(215,200,20,60,'Cancel'),
text(10,20,64,310,'Input the object name then select a property which describes the object (deselect all
properties to remove a property from an object. Clicking SAVE records the change:'),
text(90,20,16,60,'Object:'),
edit(110,20,16,100,'',gread(Object)),
text(90,165,16,130,'Object Categories'),
menu(106,145,50,180,[different_object_types,resource,resource_container,recepticable],[different_object_t
ypes],Plist),
text(135,20,16,100,'Objects are:'),
text(155,20,16,100,O1),
text(171,20,16,100,O2),
text(187,20,16,100,O3),
text(203,20,16,100,O4),
],Button,property_check(Object,Plist)),
save_properties(Object,Plist).

/* Several rules to permit appropriate recording\changing of given
   object properties. */

save_properties(Object,Plist) :-
Plist=[Prop|Rest],not target_pdata(Object,_),
assertz(target_pdata(Object,Prop)),!.

save_properties(Object,Plist) :-
target_pdata(Object,_),Plist=[],
retract(target_pdata(Object,_)),!.

save_properties(Object,Plist) :-
target_pdata(Object,_),Plist=[Prop|Rest],
retract(target_pdata(Object,_)),
assertz(target_pdata(Object,Prop)),!.

/* Rule to constrain number of objects with properties in the system. */

property_check(D,B,Object,Plist) :-
Plist=[Prop|Rest],not target_pdata(Object,Prop),
findall(Obj,target_pdata(Obj,_),Tlist),length(Tlist,T),T = 2,
beep(60), message(['You cannot enter any more new objects with properties to Ira. Delete existing properties
first']),!,fail.

/* Rules to control input data, to validate and maintain consistency. There
   are several combinations of possibilities here, depending upon whether
   the object selects any menu selection or not. */

property_check(D,B,Object,_) :-
Object  = 'end_of_file',
```

```
beep(60), message(['You must enter the name of the object']),!,fail.

property_check(D,B,Object,_) :-
not target_object(Object),
beep(60),message(['Enter an object which is known to Ira']),!,fail.

property_check(D,B,Object,Plist) :-
Object='end_of_file',Plist=[],
beep(60),message(['Please enter the object name and select a property for the object']),!,fail.

property_check(D,B,_,Plist) :-
length(Plist,L),L>1,
beep(60),message(['Please only select one property from the menu']),!,fail.

property_check(D,B,_,_) :- !.

/* Routine to pass control to the next window */

pass_conditions(double,Win) :-
conditions_window('Conditions Window').
```

```
/* The program to elicit system requirements from the analyst - it leans
   heavily on techniques used in the conditions elicitation program */

/* Window definition */

requirements_window('Requirements Window') :-
wgcreate('Requirements Window',40,0,440,570,70,0,0,1,0),
setup_winG('Requirements Window'),
gviewer('Requirements Window',off),
wfront('Requirements Window').

setup_winG(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
present_reqt(Reqts),
add_tools(Win,[
requirements(textbox('Chicago',12,0,4,0,32,32,1,'Enter Require- ments')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_scope(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinG,[
box(25,5,145,260),
box(175,5,245,260),
box(25,270,225,210),
box(268,270,102,210),
textline('Times',14,1,5,100,'Identifying System Requirements'),

textline('Times',12,1,30,40,'What are System Requirements'),
textbox('Times',12,0,45,10,48,250,0,'The previous windows elicited basic facts about your domain. This
window and those which follow identify how these facts relates to the information system to be
implemented.'),
textbox('Times',12,0,99,10,24,250,0,'This window requests you to identify requirements of this system
(see What are Requirements).'),
textbox('Times',12,0,129,10,36,250,0,'Double click ENTER REQUIREMENTS, then select the required
knowledge state and an optional value (see below).'),

textline('Times',12,1,180,90,'Requirements'),
textbox('Times',12,0,195,10,36,250,0,'Requirements are defined as states which the information system
attempts to achieve. These states are described by object relations input previously.'),
textbox('Times',12,0,237,10,12,250,0,'States can also be specially defined:'),
textline('Times',12,0,249,10,'*'),
textline('Times',12,0,297,10,'*'),
textline('Times',12,0,333,10,'*'),
textline('Times',12,0,393,10,'*'),
textbox('Times',12,0,249,15,48,245,0,'MINIMUM_QTY implies that the system should attempt to always
hold a minimum quantity of objects, for example a stock bin contains a minimum quantity of items,'),
textbox('Times',12,0,297,15,36,245,0,'MAXIMUM_QTY implies that the system should attempt to always
hold a maximal quantity of objects,'),
textbox('Times',12,0,333,15,60,245,0,'SAME_PROPERTIES implies both objects described in the
knowledge state should have the same properties or values, for example a theatre-goer should be allocated to
a seat meeting his needs, i.e. both are less than >£20,'),
textbox('Times',12,0,393,15,24,245,0,'DATE_LIMIT implies that a state only occurs for a given length of
time.'),

textline('Times',12,1,30,300,'What are Requirements'),
textbox('Times',12,0,45,275,60,200,0,'Requirements are high-level goals for the information system. First
think of states which your system is attempting to attain, then describe these requirements in
object-relationship terms.'),
textbox('Times',12,0,111,275,72,200,0,'Requirements states can be defined in more detail by allocating
```

```
requirement types. These types describe specific details of a requirement. Requirements are shown in the bottom-left box in this window.'),
textbox('Times',12,0,189,275,24,200,0,'Ira tentatively suggests the following requirement:'),
textbox('Times',12,1,219,275,24,200,1,'Reqts'),

textline('Times',12,1,273,320,'Personnel Example'),
textbox('Times',12,0,288,275,72,200,0,'The personnel system monitors the movement of staff into and out of the organisation, so it has no specific domain states which it must help to achieve. This demonstrates the importance of purpose on identifying the scope of the domain.')]),
wkill('Conditions Window'),
enable_item('Other Inputs','Add Reqt'),
enable_item('Other Inputs','Del Reqt').

/* Subroutine necessary to guess at a possible requirement state for the
   problem domain. */

present_reqt(P) :-
get_prop(acp,selection1,Acp),
acp_reqt(O1,O2,R,Acp),
concat('Object ',R,A),
concat(A,' Object',P),!.

present_reqt(P) :-
get_prop(acp,selection1,Acp),
acp_reqt(O1,O2,R,Q,Acp),
concat('Object ',R,A),
concat(A,' Object',B),
concat(B,' when ',C),
concat(C,Q,P),!.

present_reqt('Ira is uncertain of requirements') :- !.

/* This program describes the program to elicit the static knowledge
   structure and value to identify the requirment. Note the need to delete
   properties when there exists partially-completed transactions, so done
   at beginning of each dialogue rule */

requirements(double,Win) :-
del_prop(reqt,o1),
del_prop(reqt,o2),
del_prop(reqt,rel),
del_prop(reqt,val),
mdialog(48,78,250,240,
[button(220,10,20,150,'Create Requirements'),
button(220,170,20,60,'Cancel'),
text(10,10,112,220,'Use each button to call a menu from which to select the required knowledge or the value which controls that knowledge state. You do not need to select a value before clicking CREATE:'),
button(140,50,20,150,'Object-Relation'),
button(170,50,20,150,'Requirement Type'),
],Button,requirement_menu),
assert_reqts.

/* Additional rule required to assert requirements, since they can be
   generated with or without a value */

assert_reqts :-
get_prop(reqt,o1,O1),
get_prop(reqt,o2,O2),
get_prop(reqt,rel,R),
```

```
get_prop(reqt,val,Value),
assertz(target_reqt(O1,O2,R,Value)),!.

assert_reqts :-
get_prop(reqt,o1,O1),
get_prop(reqt,o2,O2),
get_prop(reqt,rel,R),
not get_prop(reqt,val,Value),
assertz(target_reqt(O1,O2,R)).

/* Sub-windows containing windows for the three options in the main
  window */

requirement_menu(D,4) :- !,
requirement_menu1(O1,O2,R),
set_prop(reqt,o1,O1),
set_prop(reqt,o2,O2),
set_prop(reqt,rel,R),fail.

requirement_menu(D,5) :- !,
requirement_menu2(Value),
set_prop(reqt,val,Value),fail.

requirement_menu(D,B) :-
findall(O1,target_reqt(O1,_,_),List1),
findall(O2,target_reqt(O2,_,_,_),List2),
length(List1,L1),length(List2,L2),
L=L1+L2,L>=2,
beep(60),message(['You have already entered two requirements - delete existing requirements']),!,fail.

requirement_menu(D,B) :-
get_prop(reqt,o1,O1),
get_prop(reqt,o2,O2),
get_prop(reqt,rel,R),
not get_prop(reqt,val,Value),
target_reqt(O1,O2,R),
beep(60),message(['This requirement is already known to Ira']),!,fail.

requirement_menu(D,B) :-
get_prop(reqt,o1,O1),
get_prop(reqt,o2,O2),
get_prop(reqt,rel,R),
get_prop(reqt,val,Value),
target_reqt(O1,O2,R,Value),
beep(60),message(['This requirement is already known to Ira']),!,fail.

requirement_menu(D,B) :-
not get_prop(reqt,o1,O1),
beep(60),message(['You must select an object-relation']),!,fail.
requirement_menu(D,1) :- !.

/* This first window offers a menu of the movements. To display
  movements on the list it is necessary to concat data into single data
  atoms: 1) concat each data, then 2) use findall to put this data in the
  list. get_ddata puts the target_ddata in the right format */

/* Rule for eliciting the static knowledge - it calls on the standard
  program for putting fields in scroll menus then matching on that
  selection. An option program allows for the possibility of no target
```

```
structures being input. */

requirement_menu1(Object1,Object2,Relation) :-
not target_sdata(_,_,_),beep(60),
mdialog(250,300,200,300,
[button(170,100,20,100,'Continue'),
text(10,10,96,280,'You have not yet input any object-relations from which to select. A requirement cannot
be created until the object-relations has been input to the system.')],Btn),!.

requirement_menu1(Object1,Object2,Relation) :-
findall(Data,get_sdata(Data),Datalist),
Datalist = [First|Rest],
mdialog(250,300,200,300,
[button(170,30,20,60,'Ok'),
button(170,210,20,60,'Cancel'),
text(10,10,32,280,'Select the object-relation which must be achieved by the computer system:'),
menu(60,30,98,240,Datalist,[First],Slist)],Btn,check_reqts(Slist)),
rmenu1(Slist,Object1,Object2,Relation).

rmenu1(Slist,Object1,Object2,Relation) :-
Slist = [S1|Rest],
find_sdata(Object1,Object2,Relation,S1).

check_reqts(D,B,List) :-
length(List,Total),Total =\= 1,
beep(60),message(['You must select one object-relation']),!,fail.
check_reqts(D,B,_) :- !.

/* Program to display and elicit the values for a selection */

requirement_menu2(Value) :-
mdialog(250,300,200,200,
[button(170,20,20,60,'Ok'),
button(170,120,20,60,'Cancel'),
text(10,10,48,180,'Select the appropriate requirement type for the object-relation:'),
menu(80,30,66,140,[minimum_qty,maximum_qty,same_properties,date_limit],[minimum_qty],Vlist)],Btn,
check_rvalue(Vlist)),
Vlist = [Value|Rest].

check_rvalue(D,B,Vlist) :-
length(Vlist,L),L=\=1,
beep(60),message(['You must select only one requirement type']),!,fail.

check_rvalue(D,B,Vlist) :- !.

/* Routine to pass control to the next window */

pass_scope(double,Win) :-
scope_window('Scope Window').
```

```
/* This program rolls back the search when necessary after a one partial
   or two partial match condition has been fired and the target description
   is altered. In cases where the analyst undermines the previous match
   by changing the matched target descriptions, the tool must halt
   searching & either:
   1- roll back the search (i.e. delete mappings and start again), or
   2- delete mappings and stop the searching process.
   The analyst is offered this choice and selects via a dialogue buttons.
   This program also requires a careful match between target data and
   recorded analogous matches. */

/* Dialogue to offer the analyst the choice of halting or restarting the
   search process. */

rollback_dialogue :-
beep(60),mdialog(85,100,230,350,
[button(200,20,20,120,'Search Again'),
text(20,20,160,310,'You have modified some facts about the new system which were critical to identifying
the existing analogous match. Ira will have to undo these matches and begin the search again:'),
button(200,270,20,60,'Halt'),
],Btn,rollback_buttons).

rollback_buttons(D,1) :-
removeall_mappings,
searching_acps(top),!.

rollback_buttons(D,3) :-
removeall_mappings,!.

/* The top-level rule included in partial match windows to identify
   whether the target domain supporting analogous mappings has been
   modified. */

check_targetchange(Win) :-
changed_target,wkill(Win),
rollback_dialogue,!.

check_targetchange(Win) :- !.

/* Second level rule which identifies changes to the target facts which
   support analogous mappings. */

changed_target :-
rec_statmapping(Tobj1,Tobj2,_,_,Relation,_,_),
not target_sdesc(Tobj1,Tobj2,Relation),!.

changed_target :-
rec_dynmapping(Tobj1,Tobj2,Tobj3,_,_,_,Relation,_,_),
not target_ddesc(_,Tobj1,Tobj2,Tobj3,Relation),!.

changed_target :-
rec_propmapping(Tobj1,_,Property,_),
not target_pdata(Tobj1,Property),!.

changed_target :-
rec_condmapping(Tobj1,Tobj2,Tobj3,Rel1,Tobj4,Tobj5,Rel2,
_,_,_,_,_,_,Condition,_),
not target_cdata(Tobj1,Tobj2,Tobj3,Rel1,Tobj4,Tobj5,Rel2,Condition),!.
```

```
changed_target :-
rec_reqtmapping1(Tobj1,Tobj2,_,_,Relation,_),
not target_reqt(Tobj1,Tobj2,Relation),!.

changed_target :-
rec_reqtmapping2(Tobj1,Tobj2,_,_,Relation,Property,_),
not target_reqt(Tobj1,Tobj2,Relation,Property),!.

changed_target :-
rec_scopemapping(Tobj1,Tobj2,Tobj3,_,_,_,Relation,_),
not target_scope(Tobj1,Tobj2,Tobj3,Relation),!.

changed_target :-
rec_labelmapping(Label,_),
not target_label(Label),!.

changed_target :-
rec_phymapping(Tobj1,_,Property,_),
not target_phyprop(Tobj1,Property),!.
```

```
/* The program to elicit the system scope from the analyst. It is simpler
   than previous programs in that it only requires one window ans scroll
   menu */

/* Window definition */

scope_window('Scope Window') :-
wgcreate('Scope Window',40,0,440,570,70,0,0,1,0),
setup_winH('Scope Window'),
gviewer('Scope Window',off),
wfront('Scope Window').

setup_winH(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
present_scope(Prescope),
add_tools(Win,[
scope(textbox('Chicago',12,0,4,0,32,32,1,'Enter System Scope')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_label(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinH,[
box(25,5,170,260),
box(200,5,220,260),
box(25,270,135,210),
box(165,270,220,210),
textline('Times',14,1,5,50,'Identifying the Scope of the Information System'),

textline('Times',12,1,30,40,'What is the Scope of the System'),
textbox('Times',12,0,45,10,36,250,0,'This window encourages you to identify the scope of control of the
information system. This is done by examining each function.'),
textbox('Times',12,0,87,10,36,250,0,'Each function can be described as either initiated by the information
system or initiated by other events beyond the information system.'),
textbox('Times',12,0,129,10,48,250,0,'To enter a function not initiated by the information system double
click ENTER SYSTEM SCOPE then enter the function not initiated by the system.'),

textline('Times',12,1,205,75,'System Scope'),
textbox('Times',12,0,220,10,12,250,0,'Each function is initiated either by:'),
textline('Times',12,0,238,10,'*'),
textline('Times',12,0,250,10,'*'),
textbox('Times',12,0,238,20,24,240,0,'the information system, or'),
textbox('Times',12,0,250,20,24,240,0,'events outside the information system.'),
textbox('Times',12,0,268,10,48,250,0,'Many events are initiated by the information system, for example
the allocation of engine drivers to trains at the beginning of every shift. These events are not beyond the
control of the information system.'),
textbox('Times',12,0,322,10,48,250,0,'Other functions only occur when events in the outside world cause
them to occur. These functions are beyond the scope of the information system and should be identified in
this window.'),

textline('Times',12,1,30,340,'Hints'),
textline('Times',12,0,45,275,'*'),
textbox('Times',12,0,45,285,36,190,0,'Think of your domain in physical terms (see example below) and
sketch it out beforehand.'),
textline('Times',12,0,87,275,'*'),
textbox('Times',12,0,87,285,12,190,0,'Consider each function in turn.'),
textline('Times',12,0,105,275,'*'),
textbox('Times',12,0,105,285,36,190,0,'Ira tentatively suggests that the following function may be beyond
the system scope:'),
textline('Times',12,1,146,295,Prescope),
```

```
textline('Times',12,1,170,320,'Personnel Example'),
box(200,280,80,185),
textline('Times',12,2,203,420,'World'),
speckled(fillcircle(240,305,20)),
speckled(fillcircle(240,425,22)),
fillbox(230,295,10,10),
fillbox(241,306,10,10),
fillbox(222,420,10,10),
fillbox(243,412,10,10),
fillbox(230,429,10,10),
fillbox(248,427,10,10),
textline('Times',12,2,260,395,'Organisation'),
textline('Times',12,2,260,282,'Agency'),
line((240,330),(240,350)),
line((235,345),(240,350)),
line((245,345),(240,350)),
fillbox(235,360,10,10),
line((240,375),(240,400)),
line((235,395),(240,400)),
line((245,395),(240,400)),
textline('Times',12,2,245,350,'Many'),
textline('Times',12,2,223,350,'Staff'),

textbox('Times',12,0,300,275,72,200,0,'The personnel system is a monitoring system and does not affect
staff joining the organisation. As a result movement of staff into the organisation is beyond the scope of the
information system, and should be identified as such.')]),
wkill('Requirements Window'),
enable_item('Other Inputs','Add Scope'),
enable_item('Other Inputs','Del Scope').

/* Subroutine needed to approximate the possible scope of the current
   system. */

present_scope(Function) :-
get_prop(acp,selection,Acp),
acp_scope(Function,Acp),!.

present_scope('Ira is uncertain of functions').


/* This program describes the program to elicit the dynamic knowledge
   structure and value to identify the scope. Note the need to delete
   properties when there exists partially-completed transactions, so done
   at beginning of each dialogue rule. There is also a control version of the
   rule included if there are no input object movements. */

scope(double,Win) :-
not target_ddata(_,_,_,_),
mdialog(48,78,160,260,
[button(130,80,20,100,'Continue'),
text(10,10,112,240,'You have not yet input any object movements which may be used to identify the scope
of the required computer system. Please input object movements before identifying the system
scope.')],Btn),!.

scope(double,Win) :-
findall(Mvmt,target_ddata(Mvmt,_,_,_),Datalist),
Datalist = [First|Rest],
mdialog(48,75,270,260,
```

```
[button(240,10,20,150,'Create Scope'),
button(240,190,20,60,'Cancel'),
text(10,10,112,240,'Select object movements which are beyond the control of the computer system. You
may select more than one movement, however only one movement can be selected at a time before clicking
CREATE:'),
menu(130,10,98,240,Datalist,[First],Mlist)],
Btn,scope_menu(Mlist)),
Mlist = [Movement],
assertz(target_scope(Movement)).

scope_menu(D,B,Mlist) :-
length(Mlist,Total),Total =\= 1,
beep(60),message(['You must select one function from the scroll menu']),!,fail.

scope_menu(D,B,_) :-
findall(M,target_scope(M),Mlist),
length(Mlist,L),L>=2,
beep(60),message(['You have already entered 2 functions beyond the scope of the information
system']),!,fail.

scope_menu(D,B,Mlist) :-
Mlist=[M|Rest],target_scope(M),
beep(60),message(['This function beyond the system scope has already been entered into Ira']),!,fail.

scope_menu(D,B,_) :- !.

/* Routine to pass control to the next window */

pass_label(double,Win) :-
label_window('Label Window').
```

```
/* This program is quite complex, and attempts to elicit the static
   structural relations. It includes a scrolling menu from which to select
   the most appropriate structural relation.

   It also has two windows created for the two phases in the development
   of structural relations: (i) elicitation of structures pertaining to specific
   state tranistions, (ii) structures added beyond the scope of these
   transitions, i.e. WORLD and additional structures. The first is called
   the STRUCTURAL window and the second is called the STRUCTURES
   window. However, both windows call the same dialogue routines. The
   windows are differentiated by the contents of the windows, which
   directs the analysts to input different types of structural relations. */

/* First STRUCTURAL window definition. This window includes the simple
   five-line routine to remove the current function from the function list
   ready for the next cycle of the function definition process. */

structural_window('Structural Window') :-
wgcreate('Structural Window',40,0,440,570,70,0,0,1,0),
setup_winCa('Structural Window'),
gviewer('Structural Window',off),
wfront('Structural Window').

setup_winCa(Win) :-
get_prop(function,list,List),
List=[Func|Rest],
prepare_line1(Func,Line1),
prepare_line2(Func,Line2),
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
contains_relations(textbox('Chicago',12,0,4,0,32,32,1,'Enter Stru- cture')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
stop_addfn(textbox('Chicago',12,0,4,0,32,32,1,'Restart Function Input')),
pass_funcontrol(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinCa,[
box(25,5,181,350),
box(234,5,186,260),
box(215,270,175,210),
textline('Times',14,1,5,100,'Identifying the Structure of Objects'),

textbox('Times',12,0,30,10,36,340,0,'Use this window to identify and input the structure of objects which
were identified in the previous window. This is done by expressing the relationship between pairs of these
objects.'),
textbox('Times',12,0,78,10,36,340,0,'Ira suggests that, in this instance, you input the following two
structural relations (definitions of these relations are defined in more detail below):'),
textbox('Chicago',12,0,120,10,10,340,1,Line1),
textbox('Chicago',12,0,135,10,10,340,1,Line2),
textbox('Times',12,0,165,10,36,340,0,'To enter an object structure double click ENTER STRUCTURE
then identify two objects and select a relationship between them. Possible relationships are described
below.'),

textline('Times',12,1,239,20,'Types of relationships between objects'),
textbox('Times',12,0,254,10,24,250,0,'Two relationships are available to describe the link between
objects:'),
textline('Times',12,0,286,30,'* X contains_one Y'),
textline('Times',12,0,286,130,'* X contains_many Y'),
textbox('Times',12,0,304,10,48,250,0,'These relationships represent the structure of objects described in
```

```
your sketch of the function. They detail the relationship between objects in the sketch.'),
textbox('Times',12,0,346,10,60,250,0,'At any moment in time X may contain one or many Y, which must
be specified by the selected relationship. If X may only even contain one Y then select Contains _one,
otherwise select Contains _many.'),

textline('Times',12,1,220,310,'Personnel Example'),
speckled(fillcircle(260,325,20)),
speckled(fillcircle(260,400,22)),
fillbox(250,315,10,10),
fillbox(261,326,10,10),
fillbox(242,390,10,10),
fillbox(263,382,10,10),
fillbox(250,409,10,10),
fillbox(263,405,10,10),
textline('Times',12,2,280,365,'Organisation'),
textline('Times',12,2,280,302,'Agency'),

textline('Times',12,0,302,275,'Appropriate structural relations are:'),
textline('Times',12,0,320,275,'*  organisation contains_many staff,'),
textline('Times',12,0,332,275,'*  agency contains_many staff.'),
textbox('Times',12,0,350,275,36,200,0,'That is, at any time, the agency can contain many staff and the
organisation can also contain many staff.')]),
wkill('Function Definition Window'),
enable_item('Objects','Add Structure'),
enable_item('Objects','Del Structure'),
get_prop(function,list,List),
List=[Func|Rest],
remove(Func,List,Newlist),
del_prop(function,list),
set_prop(function,list,Newlist).

/* Two simple subroutines to prepare the two Prompt lines to improve the
   final display & usability of the system. */

prepare_line1(Func,Line) :-
target_ddata(Func,Object,Source,Destination,_),
concat(Source,' contains_one/many ',A),
concat(A,Object,Line).

prepare_line2(Func,Line) :-
target_ddata(Func,Object,Source,Destination,_),
concat(Destination,' contains_one/many ',A),
concat(A,Object,Line).

/* Separate input dialogue window, to permit more control over the
   objects displayed for selection, and to support tighter control over the
   relations offered to users. Apart from this, it is the same as the static
   relations dialogue, using the same error controls, etc. */

contains_relations(double,Win) :-
build_objects(Objlist),
remove(world,Objlist,Olist),
Olist = [Object1,Object2,Object3,Object4],
mdialog(48,78,280,370,
[button(250,60,20,100,'Create'),
button(250,240,20,100,'Cancel'),
text(10,10,80,350,'Please choose two objects then select the most appropriate relation between them. To
generate the relation between the objects click the CREATE button:'),
text(90,119,16,135,'Structural Relations'),
```

```
text(90,10,16,85,'First Object:'),
text(90,260,16,100,'Second Object:'),
edit(130,10,16,100,'',gread(ObjectA)),
edit(130,260,16,100,'',gread(ObjectB)),
menu(118,125,34,120,['contains_one','contains_many'],['contains_one'],Relation),
text(170,70,16,190,'Objects to choose from are:'),
text(170,260,16,100,Object1),
text(185,260,16,100,Object2),
text(201,260,16,100,Object3),
text(217,260,16,100,Object4),
],Button,static_check(Relation,ObjectA,ObjectB)),
Relation = [Rel|Rest],
assertz(target_sdata(ObjectA,ObjectB,Rel)).

/* Second STRUCTURES window definition. */

structures_window('Structures Window') :-
wgcreate('Structures Window',40,0,440,570,70,0,0,1,0),
setup_winCb('Structures Window'),
gviewer('Structures Window',off),
wfront('Structures Window').

setup_winCb(Win) :-
unheaded_objects(Obj1,Obj2,Obj3,Obj4),
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
static_relations(textbox('Chicago',12,0,4,0,32,32,1,'Enter Stru- cture')),
general_help(textbox('Chicago',12,0,6,0,32,32,1,'General Help')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem')),
pass_properties(textbox('Chicago',12,0,6,0,32,32,1,'Next Window'))],1),
add_pic(Win,picwinCb,[
box(25,5,187,260),
box(217,5,203,260),
box(25,270,205,210),
box(235,270,185,210),
textline('Times',14,1,5,90,'Identifying further Structures between Objects'),

textline('Times',12,1,30,45,'Relationships between Objects'),
textbox('Times',12,0,45,10,36,250,0,'Your sketch of system functions can be developed further to describe
more of the problem domain. Develop your sketch in the following ways:'),
textline('Times',12,0,87,10,'* combine function sketches into a single diagram,'),
textline('Times',12,0,99,10,'* draw a boundary (WORLD) around this diagram,'),
textline('Times',12,0,111,10,'* add addtional structures to the diagram,'),
textbox('Times',12,0,129,10,48,250,0,'Each of these changes to your sketch can be described by an
entity-relationship which should be input into Ira. See the remainder of this window for details.'),
textbox('Times',12,0,183,10,12,250,0,'Enter relationships into Ira as before.'),

textline('Times',12,1,222,20,'Types of relationships between objects'),
textbox('Times',12,0,237,10,32,250,0,'Four relationships are available to describe the link between two
objects:'),
textline('Times',12,0,261,30,'* has_one'),
textline('Times',12,0,273,30,'* has_many'),
textline('Times',12,0,261,130,'* contains_one'),
textline('Times',12,0,273,130,'* contains_many'),
textbox('Times',12,0,291,10,48,250,0,'Has_one and Has_many relations represent static relationships
between objects which never change. They describe relationships between objects which are not processed
by system functions.'),
textbox('Times',12,0,345,10,24,250,0,'Contains_one & Contains_many relations were described in the
```

```
previous window.'),
textbox('Times',12,0,375,10,36,250,0,'You should use these 4 relationships to describe new structures
created by linking function structures and adding the domain boundary (see World entity).'),

textline('Times',12,1,30,310,'The WORLD entity'),
textbox('Times',12,0,45,275,60,190,0,'The WORLD entity represents the entire domain. It is shown
graphically on your sketch by the boundary around your diagram, and has been automatically created by
Ira.'),
textbox('Times',12,0,111,275,24,190,0,'Ira recommends that you enter the following mappings:'),
textline('Chicago',12,0,141,275,'world has_one/many:'),
textline('Chicago',12,0,156,390,Obj1),
textline('Chicago',12,0,171,390,Obj2),
textline('Chicago',12,0,186,390,Obj3),
textline('Chicago',12,0,201,390,Obj4),

textline('Times',12,1,240,320,'Personnel Example'),
box(255,290,65,185),
speckled(fillcircle(280,335,20)),
speckled(fillcircle(280,410,22)),
fillbox(270,325,10,10),
fillbox(281,336,10,10),
fillbox(262,400,10,10),
fillbox(273,392,10,10),
fillbox(268,414,10,10),
fillbox(283,410,10,10),
textline('Times',12,2,300,375,'Organisation'),
textline('Times',12,2,300,312,'Agency'),
textline('Times',12,2,260,440,'World'),

textbox('Times',12,0,325,275,24,200,0,'The domain boundary was added to the record function, so:'),
textline('Times',12,0,349,275,'*  world has_one organisation,'),
textline('Times',12,0,361,275,'*  world has_one agency.'),
textbox('Times',12,0,379,275,36,200,0,'The personnel domain has one organisation and one agency, so it
can be represented and described quite simply.')]),
wkill('Structural Window'),
enable_menu('Objects'),
enable_item('Objects','Add Function'),
enable_item('Objects','Del Function'),
enable_item('Objects','Add Extra Object'),
enable_item('Objects','Delete Extra Object'),
enable_item('Objects','Mod Object').

/* The analysis routine to identify objects not yet connected to the
   hierarchical structural model via the WORLD entity. There can only be
   a maximum number of 4 such entities, due to constraints on number of
   objects and functional structuring. This is done is the following way
   (since difficult to fire rules which identify when they do not occur):
   - develop list of all objects, then exclude world from it,
   - develop list of object which fit into the hierarchical structure,
   - remove these objects from original list to identify the unlinked
     entities,
   - fill out list with spaces to give it ability to easily dump it on the
     screen. */

unheaded_objects(Obj1,Obj2,Obj3,Obj4) :-
findall(Allobjects,target_object(Allobjects),Allist),
remove(world,Allist,Newlist),
findall(Okobjects,target_sdata(_,Okobjects,_),Oklist),
remove_all(Oklist,Newlist,Finalist),
```

```
space_objects(Finalist,Obj1,Obj2,Obj3,Obj4).

space_objects(Finalist,Obj1,Obj2,Obj3,Obj4) :-
length(Finalist,0),Obj1='',Obj2='',Obj3='',Obj4='',!.

space_objects(Finalist,Obj1,Obj2,Obj3,Obj4) :-
length(Finalist,1),Finalist=[Obj1],Obj2='',Obj3='',Obj4='',!.

space_objects(Finalist,Obj1,Obj2,Obj3,Obj4) :-
length(Finalist,2),Finalist=[Obj1,Obj2],Obj3='',Obj4='',!.

space_objects(Finalist,Obj1,Obj2,Obj3,Obj4) :-
length(Finalist,3),Finalist=[Obj1,Obj2,Obj3],Obj4='',!.

space_objects(Finalist,Obj1,Obj2,Obj3,Obj4) :-
length(Finalist,4),Finalist=[Obj1,Obj2,Obj3,Obj4],!.

/* This program describes the program to elicit static structural relations
   to describe the new target problem */

static_relations(double,Win) :-
build_objects(Objlist),
Objlist = [Object1,Object2,Object3,Object4,Object5],
mdialog(48,78,310,370,
[button(280,60,20,100,'Create'),
button(280,240,20,100,'Cancel'),
text(10,10,80,350,'Please choose two objects then select the most appropriate relation between them. To
generate the relation between the objects click the CREATE button:'),
text(90,119,16,135,'Structural Relations'),
text(90,10,16,85,'First Object:'),
text(90,260,16,100,'Second Object:'),
edit(130,10,16,100,'',gread(ObjectA)),
edit(130,260,16,100,'',gread(ObjectB)),
menu(110,125,66,120,['has_one','has_many','contains_one','contains_many'],['has_one'],Relation),
text(186,70,16,190,'Objects to choose from are:'),
text(186,260,16,100,Object1),
text(201,260,16,100,Object2),
text(217,260,16,100,Object3),
text(233,260,16,100,Object4),
text(249,260,16,100,Object5),
],Button,static_check(Relation,ObjectA,ObjectB)),
Relation = [Rel|Rest],
assertz(target_sdata(ObjectA,ObjectB,Rel)).

/* Rule to constrain the number of static relations to eight static structure.
  This is a simple input validation mechanism based on desire not to have
  more than two good matches for a structure. */

static_check(D,B,_,_,_) :-
findall(Rel,target_sdata(_,_,Rel),Tlist),
length(Tlist,T),T = 8,
beep(60), message(['You cannot enter any more enter-relations. Delete existing relations first']),!,fail.

/* Rules to control input data, to validate and maintain consistency.
  Consistency is checked within the applciation of relations between
  a specific set of objects - i.e. mutual exclusion between 'has_no',
  'has_one' and 'has_many'.*/

static_check(D,B,_,Obj1,_) :-
```

```
Obj1  = 'end_of_file',
beep(60), message(['You must enter the name of an object']),!,fail.

static_check(D,B,_,_,Obj2) :-
Obj2  = 'end_of_file',
beep(60), message(['You must enter the name of an object']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
target_sdata(Obj1,Obj2,Rel),
beep(60),message(['I am sorry but this entity-relation is already known to Ira']),!,fail.

/* Sequence of rules to check the program consistency within static
   structure. The first six rules identify the existence of contradictory
   structures, then these are followed are nine rules which identify
   circular definitions of data, i.e. A is-in B and B is-in A, which just
   cannot happen in the hierarchy model of the physical world */

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_no',
target_sdata(Obj1,Obj2,'has_one'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_no',
target_sdata(Obj1,Obj2,'has_many'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_one',
target_sdata(Obj1,Obj2,'has_no'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_one',
target_sdata(Obj1,Obj2,'has_many'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_many',
target_sdata(Obj1,Obj2,'has_no'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_many',
target_sdata(Obj1,Obj2,'has_one'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations']),!,fail.

/* Circular definition consistency rules (9 rules - 3x3 relations) */

/* Testing the 'has-no' structure */

static_check(D,B,Relations,Obj1,Obj2) :-
```

```
Relations = [Rel|Rest],
Rel = 'has_no',
target_sdata(Obj2,Obj1,'has_no'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_no',
target_sdata(Obj2,Obj1,'has_one'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_no',
target_sdata(Obj2,Obj1,'has_many'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

/* Testing the 'has-one' structure */

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_one',
target_sdata(Obj2,Obj1,'has_no'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_one',
target_sdata(Obj2,Obj1,'has_one'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_one',
target_sdata(Obj2,Obj1,'has_many'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

/* Testing the 'has-many' structure */

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_many',
target_sdata(Obj2,Obj1,'has_no'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_one',
target_sdata(Obj2,Obj1,'has_many'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.
```

```
static_check(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
Rel = 'has_many',
target_sdata(Obj2,Obj1,'has_many'),
beep(60),message(['I am sorry but this entity-relation contradicts existing relations describing the new
system']),!,fail.

/* Remaining basic validation rules, similar to those used in other
   screeens */

static_check(D,B,Relations,Obj1,Obj2) :-
length(Relations,Total),Total = 1,
target_object(Obj1),
target_object(Obj2),!.

static_check(D,B,Relations,_,_) :-
length(Relations,Total),Total =\= 1,
beep(60), message(['Enter one relation to describe the link between the objects']),!,fail.

static_check(D,B,_,Obj1,_) :-
not target_object(Obj1),
beep(60),message(['Enter an object from the list displayed']),!,fail.

static_check(D,B,_,_,Obj2) :-
not target_object(Obj2),
beep(60),message(['Enter an object from the list displayed']),!,fail.

/* Routine to control the input and definition of functions, or the
   input of additional structural predicates at the end of the functions-
   input loop. The current function has already been deleted before this
   stage in the iteration. */

pass_funcontrol(double,Win) :-
get_prop(function,list,[]),
structures_window('Structures Window'),!.

pass_funcontrol(double,Win) :-
dynamic_window('Function Definition Window').

/* Routine to pass control to the next window */

pass_properties(double,Win) :-
properties_window('Categories Window').
```

```
/* Programs which are called by the Structure\Mvmt Menu */

/* Delete an existing static relation. This program includes two validation
   checks which prohibit the analyst from deleting a structural
   component which currently supports a system requirement or an
    identified condition, thus maintaining system consistency. */

del_structure :-
build_objects(Objlist),
Objlist = [Object1,Object2,Object3,Object4,Object5],
mdialog(60,200,310,400,
[button(280,60,20,100,'Delete'),
button(280,240,20,100,'Cancel'),
text(10,10,80,380,'Please choose two objects and the relation between them which is to be deleted, then
click the DELETE button:'),
text(90,119,16,135,'Structural Relations'),
text(90,10,16,85,'First Object:'),
text(90,260,16,100,'Second Object:'),
edit(130,10,16,100,'',gread(ObjectA)),
edit(130,260,16,100,'',gread(ObjectB)),
menu(110,125,66,120,['has_one','has_many','contains_one','contains_many'],['has_one'],Relation),
text(186,70,16,190,'Objects to choose from are:'),
text(186,260,16,100,Object1),
text(201,260,16,100,Object2),
text(217,260,16,100,Object3),
text(233,260,16,100,Object4),
text(249,260,16,100,Object5),
],Button,del_stcheck(Relation,ObjectA,ObjectB)),
Relation = [Rel|Rest],
retract(target_sdata(ObjectA,ObjectB,Rel)).

del_stcheck(D,B,_,Obj1,_) :-
Obj1  = 'end_of_file',
beep(60), message(['You must enter the name of an object']),!,fail.

del_stcheck(D,B,_,_,Obj2) :-
Obj2  = 'end_of_file',
beep(60), message(['You must enter the name of an object']),!,fail.

del_stcheck(D,B,_,Obj1,_) :-
not target_object(Obj1),
beep(60),message(['Enter an object from the list displayed']),!,fail.

del_stcheck(D,B,_,_,Obj2) :-
not target_object(Obj2),
beep(60),message(['Enter an object from the list displayed']),!,fail.

del_stcheck(D,B,Relations,_,_) :-
length(Relations,Total),Total =\= 1,
beep(60), message(['Enter one relation to describe the link between the objects']),!,fail.

del_stcheck(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
not target_sdata(Obj1,Obj2,Rel),
beep(60),message(['I am sorry but this structure is not known to Ira']),!,fail.

del_stcheck(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
target_reqt(Obj1,Obj2,Rel),
```

```
beep(60),message(['This structure cannot currently be deleted since it identifies a system
requirement']),!,fail.

del_stcheck(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
target_reqt(Obj1,Obj2,Rel,Value),
beep(60),message(['This structure cannot currently be deleted since it identifies a system
requirement']),!,fail.

del_stcheck(D,B,Relations,Obj1,Obj2) :-
Relations = [Rel|Rest],
target_cdata(_,_,_,_,Obj1,Obj2,Rel,_),
beep(60),message(['This structure cannot currently be deleted since it supports a system condition']),!,fail.

del_stcheck(D,B,_,_,_) :- !.

/* Create an existing dynamic relation representing a function. This
   dialogue must be separate from the dialogue describing functions in the
   controlled dialogue due to the inbuilt subroutines which control the
   iteration for the development of the functional definitions. */

add_movement :-
build_objects(Objlist),
Objlist = [O1,O2,O3,O4,O5],
mdialog(48,78,340,400,
[button(310,130,20,60,'Create'),
button(310,230,20,60,'Cancel'),
text(10,10,96,380,'First select a function from the selection. You should then identify the main object
processed by the function, its initial and final positions and the number of objects processed by the function.
Finally click CREATE to record this functional definition:'),
text(120,10,16,65,'Function:'),
menu(120,78,66,160,[loan,borrow,dispatch,send,lend,goods_out,receipt,input,goods_in,arrival,addition,al
locate,assign,place,connect,join,return,finish_loan,check_position,monitor,record],[loan],Flist),
text(200,10,16,120,'Processed Object:'),
edit(200,135,16,100,'',gread(Object)),
text(275,10,16,120,'Quantities Moved:'),
edit(275,135,16,100,'move_many',Relation),
text(225,80,16,55,'Initial:'),
edit(225,135,16,100,'',gread(Source)),
text(250,90,16,45,'Final:'),
edit(250,135,16,100,'',gread(Destination)),
text(120,270,16,110,'Known Objects:'),
text(145,280,16,100,O1),
text(161,280,16,100,O2),
text(177,280,16,100,O3),
text(193,280,16,100,O4),
text(209,280,16,100,O5),
],Button,addmvmt_check(Flist,Relation,Object,Source,Destination)),
Flist=[Func|Rest],
createall_objects(Object,Source,Destination),
assertz(target_ddata(Func,Object,Source,Destination,Relation)).

/* Rules to control input data, to validate and maintain consistency. */

addmvmt_check(D,B,Flist,_,_,_,_) :-
length(Flist,L),L=\=1,
beep(60), message(['You must select one function from the available menu']),!,fail.

addmvmt_check(D,B,_,_,Object,_,_) :-
```

```
Object = 'end_of_file',
beep(60), message(['You must enter the name of the object to be processed']),!,fail.

addmvmt_check(D,B,_,_,_,Source,_) :-
Source = 'end_of_file',
beep(60), message(['You must enter the name of the initial position of the object']),!,fail.

addmvmt_check(D,B,_,_,_,_,Destination) :-
Destination = 'end_of_file',
beep(60), message(['You must enter the name of the final position of the object']),!,fail.

addmvmt_check(D,B,_,Relation,_,_,_) :-
Relation = 'end_of_file',
beep(60), message(['You must enter the type of movement (move_one or move_many']),!,fail.

addmvmt_check(D,B,_,Relation,_,_,_) :-
Relation=\= 'move_one',Relation=\='move_many',
beep(60), message(['You must enter the type of movement (move_one or move_many)']),!,fail.

addmvmt_check(D,B,_,_,Object,_,_) :-
not valid_character(Object),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),!,fail.

addmvmt_check(D,B,_,_,_,Source,_) :-
not valid_character(Source),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),!,fail.

addmvmt_check(D,B,_,_,_,_,Destination) :-
not valid_character(Destination),
beep(60),message(['An object must begin with a small letter and only contain letters or numbers']),!,fail.

addmvmt_check(D,B,Flist,Relation,Object,Source,Destination) :-
Flist=[Func],target_ddata(Func,_,_,_,_),
beep(60),message(['This function has already been input into Ira. ~MPlease try again']),!,fail.

addmvmt_check(D,B,_,_,_,_,_) :-
findall(F,target_ddata(F,_,_,_,_),Flist),
length(Flist,L),L=2,
beep(60),message(['You have already entered two functions. ~MDelete existing functions first']),!,fail.

/* The following few rules ensure the consistency of the input
   describing the dynamic relation rules. Look for direction contradictions
   between rules in the same direction, then contradictions between
   opposing objects */

addmvmt_check(D,B,_,Relation,Object,Source,Destination) :-
Relation = 'move_one',
target_ddata(_,Object,Source,Destination,'move_many'),
beep(60),message(['This movement contradicts previous movement describing the new system']),!,fail.

addmvmt_check(D,B,_,Relation,Object,Source,Destination) :-
Relation = 'move_many',
target_ddata(_,Object,Source,Destination,'move_one'),
beep(60),message(['This movement contradicts previous movement describing the new system']),!,fail.

addmvmt_check(D,B,_,_,_,_,_) :- !.

/* Delete an existing dynamic relation representing a function. Since
  function is the central concept of this system, all other structures are
```

```
   dependent upon the function, so structures and objects also disappear
   when the functions go. The functions deletes all which is solely
   dependent upon the definition of the function: all deletions are
   linked to the knowledge dependency graph provided in the
   documentation. */

del_movement :-
findall(Mvmt,target_ddata(Mvmt,_,_,_,_),Datalist),
Datalist=[First|Rest],
mdialog(58,125,300,200,
[button(270,20,20,60,'Delete'),
button(270,120,20,60,'Cancel'),
text(10,10,160,180,'Please enter the name of the function to be deleted, then click the DELETE button.
Please note that deleting a function is powerful and will also delete all objects and object-structures depend
upon this function:'),
menu(180,25,66,150,Datalist,[First],Mlist)],
Button,check_dyrelations(Mlist)),
Mlist=[Func],delete_everything(Func).

/* Rules to control input data, to validate and maintain consistency. They
   include two consistency checking rules which prohibits the deletion
   of a movement if it currently supports an identified system scope or
   a system condition. */

check_dyrelations(D,B,Mlist) :-
Mlist=[],
beep(60), message(['You must enter the name of the movement to be deleted']),!,fail.

check_dyrelations(D,B,Mlist) :-
length(Mlist,T),T>1,
beep(60), message(['Please only select one movement at a time to be deleted']),!,fail.

check_dyrelations(D,B,Mlist) :- !.

/* The following series of rules control the deletion of knowledge
   dependent solely upon the function to be deleted. Basic routine for
   identifying the extent of object dependence upon the function
   structure is the first rule given below. */

other_function(Func,Object) :-
target_ddata(F1,Object,_,_,_),
Func=\=F1,!.
other_function(Func,Object) :-
target_ddata(F2,_,Object,_,_),
Func=\=F2,!.
other_function(Func,Object) :-
target_ddata(F3,_,_,Object,_),
Func=\=F3.

/* The basic rule for deleting all candidate structures. Note the subroutine
   to remove all functional-dependent structures (scope & condition). */

delete_everything(Func) :-
target_ddata(Func,O1,O2,O3,_),
findall(Func,remove_fncdepend(Func),Anylist),
delmvmt_object(Func,O1),
delmvmt_object(Func,O2),
delmvmt_object(Func,O3),
delmvmt_structure(Func,O1,O2),
```

```
delmvmt_structure(Func,O1,O3),
delmvmt_structure(Func,O2,O3),
delmvmt_structure(Func,world,O1),
delmvmt_structure(Func,world,O2),
delmvmt_structure(Func,world,O3),
retractall(target_ddata(Func,O1,O2,O3,_)).

remove_fncdepend(Func) :-
retract(target_cdata(Func,_)).
remove_fncdepend(Func) :-
retract(target_scope(Func)).
remove_fncdepend(Func) :- !.

/* Deletion of individual objects dependent only on this function. If a
   function is deleted a subroutine exists to delete all dependents on
   that function. */

delmvmt_object(Func,Object) :-
not other_function(Func,Object),
Object=\=world,
findall(Object,remove_objdepend(Object),Anylist),
retract(target_object(Object)),!.
delmvmt_object(Func,Object) :- !.

remove_objdepend(Object) :-
retract(target_pdata(Object,_)).
remove_objdepend(Object) :-
retract(target_phyprop(Object,_)).
remove_objdepend(Object) :- !.

/* Deletion of knowledge structures dependent only on this function. Note
   the subroutine to delete requirements knowledge structure depending
   on the object structures, included in the main structure delete. */

delmvmt_structure(Func,Obj1,Obj2) :-
not other_function(Func,Obj1),
not other_function(Func,Obj2),
findall(Obj1,remove_structure(Obj1,Obj2),Anylist),!.
delmvmt_structure(Func,Obj1,Obj2) :- !.

remove_structure(Obj1,Obj2) :-
retractall(target_sdata(Obj1,Obj2,_)).
remove_structure(Obj1,Obj2) :-
retractall(target_sdata(Obj2,Obj1,_)).
remove_structure(Obj1,Obj2) :-
retractall(target_reqt(Obj1,Obj2,_)).
remove_structure(Obj1,Obj2) :-
retractall(target_reqt(Obj2,Obj1,_)).
remove_structure(Obj1,Obj2) :-
retractall(target_reqt(Obj1,Obj2,_,_)).
remove_structure(Obj1,Obj2) :-
retractall(target_reqt(Obj2,Obj1,_,_)).

/* All other deletions are dependent upon these deletions occuring. */
```

# Descriptions of the Explanation Dialogues for Retrieved Domain Abstractions

```
/* This program constructs the window frames to elicit additional
   knowledge from the analyst in cases of a partial match. It creates four
   slots which exist in different positions on the window. The complexity
   of this program comes in adding the windows correctly rather than
   additional analogous matching to identify potential matches.

   The dialogue slots are determined by type of knowledge recorded
   in the slot:
   slot 1 & 2 - state transitions,
   slot 3 & 4 - domain structures.

   When dialogue for a slot is not necessary the slot dialogue is omitted,
   just leaving a space in the window. A dummy rule for each line is still
   required to permit window rule to be successful. Below there are
   two sets of diagramming rules, one for one partial match window, &
   one for two partial matches window. This is due to the different rules
   necessary to identify omitted analogous mappings with one partial
   match and several partial matches. */

/* Add_pictures1-4 describe the simpler rules for one partial match. */

add_picture1(Win,Acp) :-
no_dynamicmapping(Obj1,Obj2,Obj3,Rel,Acp),
add_pic(Win,piccy1d,[textbox('Bookman',12,0,155,35,16,100,1,Obj2)]),
add_pic(Win,piccy1e,[textbox('Bookman',12,0,155,295,16,100,1,Obj3)]),
add_pic(Win,piccy1f,[textbox('Bookman',12,0,180,165,16,100,1,Rel)]),
add_pic(Win,piccy1a,[oval(145,30,30,110)]),
add_pic(Win,piccy1b,[oval(145,290,30,110)]),
add_pic(Win,piccy1c,[line((160,140),(160,290))]),
add_pic(Win,piccy1g,[textbox('Bookman',12,0,155,165,16,100,1,Obj1)]),
add_pic(Win,piccy1h,[box(145,165,30,100)]),
add_pic(Win,piccy1i,[line((160,165),(155,160))]),
add_pic(Win,piccy1j,[line((160,165),(165,160))]),
add_pic(Win,piccy1k,[line((160,290),(155,285))]),
add_pic(Win,piccy1l,[line((160,290),(165,285))]),!.

add_picture1(Win,Acp) :-
approximate_dynamictarget(Rel,Acp),
add_pic(Win,piccy1d,[textbox('Bookman',12,0,155,35,16,100,1,'?')]),
add_pic(Win,piccy1e,[textbox('Bookman',12,0,155,295,16,100,1,'?')]),
add_pic(Win,piccy1f,[textbox('Bookman',12,0,180,165,16,100,1,Rel)]),
add_pic(Win,piccy1a,[oval(145,30,30,110)]),
add_pic(Win,piccy1b,[oval(145,290,30,110)]),
add_pic(Win,piccy1c,[line((160,140),(160,290))]),
add_pic(Win,piccy1g,[textbox('Bookman',12,0,155,165,16,100,1,'?')]),
add_pic(Win,piccy1h,[box(145,165,30,100)]),
add_pic(Win,piccy1i,[line((160,165),(155,160))]),
add_pic(Win,piccy1j,[line((160,165),(165,160))]),
add_pic(Win,piccy1k,[line((160,290),(155,285))]),
add_pic(Win,piccy1l,[line((160,290),(165,285))]),!.

add_picture1(Win,Acp) :- !.

add_picture2(Win,Acp) :-
no_dynamicmapping(Obj1,Obj2,Obj3,Rel,Acp),
add_pic(Win,piccy2d,[textbox('Bookman',12,0,230,35,16,100,1,Obj2)]),
add_pic(Win,piccy2e,[textbox('Bookman',12,0,230,295,16,100,1,Obj3)]),
add_pic(Win,piccy2f,[textbox('Bookman',12,0,255,165,16,100,1,Rel)]),
add_pic(Win,piccy2a,[oval(220,30,30,110)]),
```

```
add_pic(Win,piccy2b,[oval(220,290,30,110)]),
add_pic(Win,piccy2c,[line((235,140),(235,290))]),
add_pic(Win,piccy2g,[textbox('Bookman',12,0,230,165,16,100,1,Obj1)]),
add_pic(Win,piccy2h,[box(220,165,30,100)]),
add_pic(Win,piccy2i,[line((235,165),(230,160))]),
add_pic(Win,piccy2j,[line((235,165),(240,160))]),
add_pic(Win,piccy2k,[line((235,290),(230,285))]),
add_pic(Win,piccy2l,[line((235,290),(240,285))]),!.

add_picture2(Win,Acp) :-
approximate_dynamictarget(Rel,Acp),
add_pic(Win,piccy2d,[textbox('Bookman',12,0,230,35,16,100,1,'?')]),
add_pic(Win,piccy2e,[textbox('Bookman',12,0,230,295,16,100,1,'?')]),
add_pic(Win,piccy2f,[textbox('Bookman',12,0,255,165,16,100,1,Rel)]),
add_pic(Win,piccy2a,[oval(220,30,30,110)]),
add_pic(Win,piccy2b,[oval(220,290,30,110)]),
add_pic(Win,piccy2c,[line((235,140),(235,290))]),
add_pic(Win,piccy2g,[textbox('Bookman',12,0,230,165,16,100,1,'?')]),
add_pic(Win,piccy2h,[box(220,165,30,100)]),
add_pic(Win,piccy2i,[line((235,165),(230,160))]),
add_pic(Win,piccy2j,[line((235,165),(240,160))]),
add_pic(Win,piccy2k,[line((235,290),(230,285))]),
add_pic(Win,piccy2l,[line((235,290),(240,285))]),!.

add_picture2(Win,Acp) :- !.

add_picture3(Win,Acp) :-
no_staticmapping(Obj1,Obj2,Rel,Acp),
add_pic(Win,piccy3d,[textbox('Bookman',12,0,305,65,16,100,1,Obj1)]),
add_pic(Win,piccy3e,[textbox('Bookman',12,0,305,265,16,100,1,Obj2)]),
add_pic(Win,piccy3f,[textbox('Bookman',10,0,295,165,16,90,1,Rel)]),
add_pic(Win,piccy3a,[oval(295,60,30,110)]),
add_pic(Win,piccy3b,[oval(295,260,30,110)]),
add_pic(Win,piccy3c,[line((310,170),(310,260))]),!.

add_picture3(Win,Acp) :-
approximate_statictarget(Rel,Acp),
add_pic(Win,piccy3d,[textbox('Bookman',12,0,305,65,16,100,1,'?')]),
add_pic(Win,piccy3e,[textbox('Bookman',12,0,305,265,16,100,1,'?')]),
add_pic(Win,piccy3f,[textbox('Bookman',10,0,295,165,16,90,1,Rel)]),
add_pic(Win,piccy3a,[oval(295,60,30,110)]),
add_pic(Win,piccy3b,[oval(295,260,30,110)]),
add_pic(Win,piccy3c,[line((310,170),(310,260))]),!.

add_picture3(Win,Acp) :- !.

add_picture4(Win,Acp) :-
no_staticmapping(Obj1,Obj2,Rel,Acp),
add_pic(Win,piccy4d,[textbox('Bookman',12,0,380,65,16,100,1,Obj1)]),
add_pic(Win,piccy4e,[textbox('Bookman',12,0,380,265,16,100,1,Obj2)]),
add_pic(Win,piccy4f,[textbox('Bookman',10,0,370,165,16,90,1,Rel)]),
add_pic(Win,piccy4a,[oval(370,60,30,110)]),
add_pic(Win,piccy4b,[oval(370,260,30,110)]),
add_pic(Win,piccy4c,[line((385,170),(385,260))]),!.

add_picture4(Win,Acp) :-
approximate_statictarget(Rel,Acp),
add_pic(Win,piccy4d,[textbox('Bookman',12,0,380,65,16,100,1,'?')]),
add_pic(Win,piccy4e,[textbox('Bookman',12,0,380,265,16,100,1,'?')]),
```

```
add_pic(Win,piccy4f,[textbox('Bookman',10,0,370,165,16,90,1,Rel)]),
add_pic(Win,piccy4a,[oval(370,60,30,110)]),
add_pic(Win,piccy4b,[oval(370,260,30,110)]),
add_pic(Win,piccy4c,[line((385,170),(385,260))]),!.

add_picture4(Win,Acp) :- !.

/* Add_pictures5-8 describe drawing rules for the more complex several
   partial matches. Each rule has a get best partial match, to identify the
   partially matched ACP with best calc-score, which is pursued further
   to attempt to match it. */

add_picture5(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
no_dynamicmapping(Obj1,Obj2,Obj3,Rel,Acp),
add_pic(Win,piccy5d,[textbox('Bookman',12,0,155,35,16,100,1,Obj2)]),
add_pic(Win,piccy5e,[textbox('Bookman',12,0,155,295,16,100,1,Obj3)]),
add_pic(Win,piccy5f,[textbox('Bookman',12,0,180,165,16,100,1,Rel)]),
add_pic(Win,piccy5a,[oval(145,30,30,110)]),
add_pic(Win,piccy5b,[oval(145,290,30,110)]),
add_pic(Win,piccy5c,[line((160,140),(160,290))]),
add_pic(Win,piccy5g,[textbox('Bookman',12,0,155,165,16,100,1,Obj1)]),
add_pic(Win,piccy5h,[box(145,165,30,100)]),
add_pic(Win,piccy5i,[line((160,165),(155,160))]),
add_pic(Win,piccy5j,[line((160,165),(165,160))]),
add_pic(Win,piccy5k,[line((160,290),(155,285))]),
add_pic(Win,piccy5l,[line((160,290),(165,285))]),!.

add_picture5(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
approximate_dynamictarget(Rel,Acp),
add_pic(Win,piccy5d,[textbox('Bookman',12,0,155,35,16,100,1,'?')]),
add_pic(Win,piccy5e,[textbox('Bookman',12,0,155,295,16,100,1,'?')]),
add_pic(Win,piccy5f,[textbox('Bookman',12,0,180,165,16,100,1,Rel)]),
add_pic(Win,piccy5a,[oval(145,30,30,110)]),
add_pic(Win,piccy5b,[oval(145,290,30,110)]),
add_pic(Win,piccy5c,[line((160,140),(160,290))]),
add_pic(Win,piccy5g,[textbox('Bookman',12,0,155,165,16,100,1,'?')]),
add_pic(Win,piccy5h,[box(145,165,30,100)]),
add_pic(Win,piccy5i,[line((160,165),(155,160))]),
add_pic(Win,piccy5j,[line((160,165),(165,160))]),
add_pic(Win,piccy5k,[line((160,290),(155,285))]),
add_pic(Win,piccy5l,[line((160,290),(165,285))]),!.

add_picture5(Win,Acplist) :- !.

add_picture6(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
no_dynamicmapping(Obj1,Obj2,Obj3,Rel,Acp),
add_pic(Win,piccy6d,[textbox('Bookman',12,0,230,35,16,100,1,Obj2)]),
add_pic(Win,piccy6e,[textbox('Bookman',12,0,230,295,16,100,1,Obj3)]),
add_pic(Win,piccy6f,[textbox('Bookman',12,0,255,165,16,100,1,Rel)]),
add_pic(Win,piccy6a,[oval(220,30,30,110)]),
add_pic(Win,piccy6b,[oval(220,290,30,110)]),
add_pic(Win,piccy6c,[line((235,140),(235,290))]),
add_pic(Win,piccy6g,[textbox('Bookman',12,0,230,165,16,100,1,Obj1)]),
add_pic(Win,piccy6h,[box(220,165,30,100)]),
add_pic(Win,piccy6i,[line((235,165),(230,160))]),
add_pic(Win,piccy6j,[line((235,165),(240,160))]),
```

```
add_pic(Win,piccy6k,[line((235,290),(230,285))]),
add_pic(Win,piccy6l,[line((235,290),(240,285))]),!.

add_picture6(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
approximate_dynamictarget(Rel,Acp),
add_pic(Win,piccy6d,[textbox('Bookman',12,0,230,35,16,100,1,'?')]),
add_pic(Win,piccy6e,[textbox('Bookman',12,0,230,295,16,100,1,'?')]),
add_pic(Win,piccy6f,[textbox('Bookman',12,0,255,165,16,100,1,Rel)]),
add_pic(Win,piccy6a,[oval(220,30,30,110)]),
add_pic(Win,piccy6b,[oval(220,290,30,110)]),
add_pic(Win,piccy6c,[line((235,140),(235,290))]),
add_pic(Win,piccy6g,[textbox('Bookman',12,0,230,165,16,100,1,'?')]),
add_pic(Win,piccy6h,[box(220,165,30,100)]),
add_pic(Win,piccy6i,[line((235,165),(230,160))]),
add_pic(Win,piccy6j,[line((235,165),(240,160))]),
add_pic(Win,piccy6k,[line((235,290),(230,285))]),
add_pic(Win,piccy6l,[line((235,290),(240,285))]),!.

add_picture6(Win,Acplist) :- !.

add_picture7(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
no_staticmapping(Obj1,Obj2,Rel,Acp),
add_pic(Win,piccy7d,[textbox('Bookman',12,0,305,65,16,100,1,Obj1)]),
add_pic(Win,piccy7e,[textbox('Bookman',12,0,305,265,16,100,1,Obj2)]),
add_pic(Win,piccy7f,[textbox('Bookman',10,0,295,165,16,90,1,Rel)]),
add_pic(Win,piccy7a,[oval(295,60,30,110)]),
add_pic(Win,piccy7b,[oval(295,260,30,110)]),
add_pic(Win,piccy7c,[line((310,170),(310,260))]),!.

add_picture7(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
approximate_statictarget(Rel,Acp),
add_pic(Win,piccy7d,[textbox('Bookman',12,0,305,65,16,100,1,'?')]),
add_pic(Win,piccy7e,[textbox('Bookman',12,0,305,265,16,100,1,'?')]),
add_pic(Win,piccy7f,[textbox('Bookman',10,0,295,165,16,90,1,Rel)]),
add_pic(Win,piccy7a,[oval(295,60,30,110)]),
add_pic(Win,piccy7b,[oval(295,260,30,110)]),
add_pic(Win,piccy7c,[line((310,170),(310,260))]),!.

add_picture7(Win,Acplist) :- !.

add_picture8(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
no_staticmapping(Obj1,Obj2,Rel,Acp),
add_pic(Win,piccy8d,[textbox('Bookman',12,0,380,65,16,100,1,Obj1)]),
add_pic(Win,piccy8e,[textbox('Bookman',12,0,380,265,16,100,1,Obj2)]),
add_pic(Win,piccy8f,[textbox('Bookman',10,0,370,165,16,90,1,Rel)]),
add_pic(Win,piccy8a,[oval(370,60,30,110)]),
add_pic(Win,piccy8b,[oval(370,260,30,110)]),
add_pic(Win,piccy8c,[line((385,170),(385,260))]),!.

add_picture8(Win,Acplist) :-
getbest_partialmatch(Acplist,Acp),
approximate_statictarget(Rel,Acp),
add_pic(Win,piccy8d,[textbox('Bookman',12,0,380,65,16,100,1,'?')]),
add_pic(Win,piccy8e,[textbox('Bookman',12,0,380,265,16,100,1,'?')]),
add_pic(Win,piccy8f,[textbox('Bookman',10,0,370,165,16,90,1,Rel)]),
```

```
add_pic(Win,piccy8a,[oval(370,60,30,110)]),
add_pic(Win,piccy8b,[oval(370,260,30,110)]),
add_pic(Win,piccy8c,[line((385,170),(385,260))]),!.

add_picture8(Win,Acplist) :- !.

/* Definition of the header text to describe possible options */

add_header(Win) :-
add_pic(Win,headerwin,[
textline('Times',14,4,5,105,'Additional Domain knowledge Required'),
textbox('Times',12,0,30,15,36,450,0,'Ira has identified several possible matches with your new problem
description. However, Ira still requires some additional information about the problem in order to confirm or
reject these matches.'),
textbox('Times',12,0,72,15,48,450,0,'Ira has identified some additional features about the new problem
below which you should consider. If these or any facts about the new problem can be identified, use the
OBJECTS & OTHER INPUTS menus to input these facts, then click CONTINUE SEARCH.')]).

/* Analogous matching rules to identify possible areas for elaboration by
   the analyst with single partial match identified. Rules examine lack of
   static mappings, dynamic mappings and critical differences of
   partially-matched Acp. This section of the program is quite complex,
   and works in the following stages:
   i)  identify source (ACP) concepts which have not been matched (i.e.
       they have not got related static_mapping (gives the relations),
   ii) identify target mappings for the source objects at either end of the
       unmapped relations (from other object mappings from successful
       mappings,
   iii) propose new target structures based on these inferences.

   Separate rules exist for the static and dynamic structures. */

/* First-level rules which are called by add_picture programs. There are
   two sets of rules operating on static and dynamic mappings. A control
   is included to ensure that the rules do not fire if no static, dynamic
   or property mappings are identified. */

no_staticmapping(Tobj1,Tobj2,Rel,Acp) :-
target_structure,
acp_sdata(Sobj1,Sobj2,Rel,Acp),
not static_mapping(T1,T2,Sobj1,Sobj2,Rel,Score,Acp),
object_mappings(Sobj1,Tobj1,Acp),
object_mappings(Sobj2,Tobj2,Acp),
not already_static(Tobj1,Tobj2,Rel),
record_static(Tobj1,Tobj2,Rel),!.

no_dynamicmapping(Tobj1,Tobj2,Tobj3,Rel,Acp) :-
target_structure,
acp_ddata(_,Sobj1,Sobj2,Sobj3,Rel,Acp),
not dynamic_mapping(T1,T2,T3,Sobj1,Sobj2,Sobj3,Rel,Score,Acp),
object_mappings(Sobj1,Tobj1,Acp),
object_mappings(Sobj2,Tobj2,Acp),
object_mappings(Sobj3,Tobj3,Acp),
not already_dynamic(Tobj1,Tobj2,Tobj3,Rel),
record_dynamic(Tobj1,Tobj2,Tobj3,Rel),!.

/* Target_structure rule to ensure target structure exists. */

target_structure :-
```

```
target_sdesc(O1,O2,R),!.
target_structure :-
target_ddesc(F,O1,O2,O3,R),!.
target_structure :-
target_pdata(O,P).

/* Second-level rules employed to identify mapped objects which
   must then be combined into structure identified above. Note the check
   to ensure that the object is not checked against itself. */

object_mappings(Sobj,Tobj,Acp) :-
target_object(Tobj),
mapped_objects(Sobj,Tobj,Score,Acp),
other_objects(Sobj,Tobj,Scores,Acp),
sort(Scores,Sorted_scores,[],1),
Sorted_scores = [Other_score|Rest],
Score >= Other_score.

/* Third-level rules employed to identify Object & Other_scores.
   The basic counter uses findall for each source of object mappings (i.e.
   basic mapping processes) while other_objects counts mappings with
   all objects except the current one to ensure that only the best object
   match is retrieved. */

mapped_objects(Sobj,Tobj,Score,Acp) :-
findall(R,static_mapping(Tobj,_,Sobj,_,R,_,Acp),S1),
findall(R,static_mapping(_,Tobj,_,Sobj,R,_,Acp),S2),
findall(R,dynamic_mapping(Tobj,_,_,Sobj,_,_,R,_,Acp),S3),
findall(R,dynamic_mapping(_,Tobj,_,_,Sobj,_,R,_,Acp),S4),
findall(R,dynamic_mapping(_,_,Tobj,_,_,Sobj,R,_,Acp),S5),
findall(R,property_mapping(Tobj,Sobj,R,Acp),S6),
length(S1,T1),length(S2,T2),length(S3,T3),length(S4,T4),
length(S5,T5),length(S6,T6),
Score is T1+T2+T3+T4+T5+T6.

other_objects(Sobj,Tobj,Scores,Acp) :-
findall(Score,(target_object(Toth),mapped_objects(Sobj,Toth,Score,Acp),
Tobj=\=Toth),Scores).

/* Approximation rules to propose weaker relations when the above
   program is unable to identify missing target object structures. It works
   in a similar way, but just retrieves unmatched relations which are
   displayed on the screen and invites object structures to be input. */

approximate_dynamictarget(Rel,Acp) :-
acp_ddata(_,Sobj1,Sobj2,Sobj3,Rel,Acp),
not dynamic_mapping(_,_,_,Sobj1,Sobj2,Sobj3,Rel,Score,Acp),
not already_dynamic('?','?','?',Rel),
record_dynamic('?','?','?',Rel),!.

approximate_statictarget(Rel,Acp) :-
acp_sdata(Sobj1,Sobj2,Rel,Acp),
not static_mapping(_,_,Sobj1,Sobj2,Rel,Score,Acp),
not already_static('?','?',Rel),
record_static('?','?',Rel),!.

/* More complex rules to identify the best-matched partial matching
   when two or more matchings are identified, based on structure score,
   then select that acp as basis for describing additional requirements for
```

```
   the problem. */

getbest_partialmatch(Acplist,Selected_acp) :-
count_partialmatches(Acplist,Scorelist),
best_partialmatch(Scorelist,Selected_acp).

/* Rule to count score for each partial match, and put score with Acp
   identifier in the list. */

count_partialmatches(Acplist,Scorelist) :-
findall((Score,Acp),(calc_structure(Acp,Score),on(Acp,Acplist)),Scorelist).

/* Rule to check the contents of the Scorelist to identify best match
   based on analogous structure. */

best_partialmatch(Scorelist,Selected_acp) :-
sort(Scorelist,Newlist,[],1),
Newlist = [(Score,Selected_acp)|Rest].

/* Subroutines to avoid repetition in the dialogue definitions, and to
   record mappings to avoid repetition by checking against them. */

already_static(Tobj1,Tobj2,Rel) :-
get_prop(static,obj1,Tobj1),
get_prop(static,obj2,Tobj2),
get_prop(static,rel,Rel).

record_static(Tobj1,Tobj2,Rel) :-
del_prop(static,obj1),
del_prop(static,obj2),
del_prop(static,rel),
set_prop(static,obj1,Tobj1),
set_prop(static,obj2,Tobj2),
set_prop(static,rel,Rel).

already_dynamic(Tobj1,Tobj2,Tobj3,Rel) :-
get_prop(dynamic,obj1,Tobj1),
get_prop(dynamic,obj2,Tobj2),
get_prop(dynamic,obj3,Tobj3),
get_prop(dynamic,rel,Rel).

record_dynamic(Tobj1,Tobj2,Tobj3,Rel) :-
del_prop(dynamic,obj1),
del_prop(dynamic,obj2),
del_prop(dynamic,obj3),
del_prop(dynamic,rel),
set_prop(dynamic,obj1,Tobj1),
set_prop(dynamic,obj2,Tobj2),
set_prop(dynamic,obj3,Tobj3),
set_prop(dynamic,rel,Rel).

/* Rule inserted into windows to initially clear variables. */

clear_acpvariables :-
del_prop(static,obj1),
del_prop(static,obj2),
del_prop(static,rel),
del_prop(dynamic,obj1),
del_prop(dynamic,obj2),
```

```
del_prop(dynamic,obj3),
del_prop(dynamic,rel).
```

```
/* This program is the program fired to provide a window to explain
   and elicit a choice between two or more good matches identified
   by the analogy engine at a specific search level in the problem space. It
   provides a window to elicit the relevant data from the analyst, which
   passes relevant data to the searching goals which already exist for the
   program. It also has a subroutine to identify the best good match for
   the two good matches based on the calc-structure score. */

goodmatches_dialogue(Acplist) :-
Acplist = [Acp1,Acp2],
bestname(Acp1,Acp2,Name),
concat(Name,'.',Nametext),
acps(Acp1,Acp_name1),
acps(Acp2,Acp_name2),
mdialog(40,85,250,400,[
text(10,10,32,380,'Ira has identified two possible abstractions, although it has identified that the best
abstraction may be the'),
text(42,10,16,380,Nametext),
text(58,10,48,380,'Please use the following buttons to examine each of these options, then select the most
appropriate option below:'),
button(220,170,20,60,'Select'),
button(130,310,20,70,'Examine'),
button(160,310,20,70,'Examine'),
radio(130,20,16,285,Acp_name1,on,Acpsel1),
radio(160,20,16,285,Acp_name2,off,Acpsel2)],Btn,
valid_goodmatches(Acpsel1,Acpsel2,Acp1,Acp2)).

valid_goodmatches(D,B,Acpsel1,Acpsel2,_,_) :-
Acpsel1 = 'on',Acpsel2 = 'on',
beep(60), message(['You must choose one abstraction. ~MPlease try again']),!,fail.

valid_goodmatches(D,B,Acpsel1,Acpsel2,_,_) :-
Acpsel1 = 'off',Acpsel2 = 'off',
beep(60), message(['You must choose one abstraction. ~MPlease try again']),!,fail.

valid_goodmatches(D,5,_,_,Acp1,Acp2) :-
set_prop(reset,dialogue,manygood),
set_prop(reset,list,[Acp1,Acp2]),
fetch_explanation(Acp1),!.

valid_goodmatches(D,6,_,_,Acp1,Acp2) :-
set_prop(reset,dialogue,manygood),
set_prop(reset,list,[Acp1,Acp2]),
fetch_explanation(Acp2),!.

valid_goodmatches(D,4,Acpsel1,Acpsel2,Acp1,Acp2) :-
identify_selection(Acpsel1,Acpsel2,Acp1,Acp2,Selected_acp),
target_name(Name),
banner(record_acpmatch(Selected_acp),['Please be patient - Ira is reasoning analogously to match
the',Name,'problem'],150,110),
del_prop(best_name),searching_acps(Selected_acp).

/* Subroutine to identify the label of the selected ocp */

identify_selection(Acpsel1,Acpsel2,Acp1,Acp2,Selected_acp) :-
Acpsel1 = 'on',
Selected_acp is Acp1,!.

identify_selection(Acpsel1,Acpsel2,Acp1,Acp2,Selected_acp) :-
```

```
Acpsel2 = 'on',
Selected_acp is Acp2.

/* Subroutine to identify the actual best match from the two candidates.
   To reduce the response time of this lengthly search process it is
   necessary to save the best fit in a property value, and retrieve it
   whenever it exists. */

bestname(Acp1,Acp2,Name) :-
get_prop(best,name,Name),!.

bestname(Acp1,Acp2,Name) :-
not get_prop(best,name,Name),
calc_structure(Acp1,S1),calc_structure(Acp2,S2),
sort([(S1,Acp1),(S2,Acp2)],[(S3,Acp3),(S4,Acp4)],[],1),
givename(S3,Acp3,S4,Acp4,Name),
set_prop(best,name,Name).

givename(S3,Acp3,S4,Acp4,Name) :-
S3=S4,Name='either abstraction',!.

givename(S3,Acp3,S4,Acp4,Name) :-
acps(Acp3,Name).

/* A second version of this program is necessary to provide and describe
   partial matches after the second pass of the selection control program.
   It is based on the above program structure however the ACP selection
   screen is different and simpler to program, requiring more input from
   the analyst. */


acceptmatches_dialogue(Acplist) :-
get_fournames(Acplist,Newlist),
Newlist=[Acp1,Acp2,Acp3,Acp4],
acps(Acp1,Acp_name1),
acps(Acp2,Acp_name2),
acps(Acp3,Acp_name3),
acps(Acp4,Acp_name4),
mdialog(40,85,260,400,[
text(10,10,80,380,'Several possible abstractions of the new problem have been identified. Please enter the
abstraction identifier and EXAMINE each option to investigate it. CHOOSE the best option to select an
abstraction:'),
text(240,5,16,10,''),
button(205,270,20,60,'Examine'),
button(230,270,20,60,'Choose'),
text(100,10,16,50,Acp1),
text(100,60,16,5,''),
text(100,70,16,315,Acp_name1),
text(120,10,16,50,Acp2),
text(120,60,16,5,''),
text(120,70,16,315,Acp_name2),
text(140,10,16,50,Acp3),
text(140,60,16,5,''),
text(140,70,16,315,Acp_name3),
text(160,10,16,50,Acp4),
text(160,60,16,5,''),
text(160,70,16,315,Acp_name4),
text(219,145,16,50,'option:'),
edit(219,205,16,50,'Acp1',Acp)],Btn,
```

```
valid_acceptmatches(Acplist,Acp)).

/* Validation and control routines from the partial matches dialogue. Note
   the strange behaviour of this ruleset. If a partial match is chosen as the
   appropriate ACP then the rule must record the good match then
   display the abstraction again on the screen. */

valid_acceptmatches(D,B,Acplist,Acp) :-
not on(Acp,Acplist),
beep(60), message(['Please choose a label from the options provided']),!,fail.

valid_acceptmatches(D,3,Acplist,Acp) :-
set_prop(reset,dialogue,acceptparts),
set_prop(reset,list,Acplist),
fetch_explanation(Acp),!.

valid_acceptmatches(D,4,Acplist,Acp) :-
set_prop(reset,dialogue,finalgood),
banner(record_acpmatch(Acp),['Please be patient - Ira is recording the selected analogous match'],150,110),
fetch_explanation(Acp),!.
```

```
/* This program describes the dialogue provided to elicit additional
   knowledge from the analyst when one partial match is identified by
   the search mechanism. The dialogue is based around a window which
   permits access to menus for data input, and explanations in the window
   to identify which knowledge should be input by the analyst. */

partmatch1_dialogue(Acp) :-
set_prop(part,match,Acp),
install_menu('Objects',['Add Object;Mod Object;Del Object;Change Properties;Add Structure;Del
Structure;Add Movement;Del Movement']),
install_menu('Other Inputs',['Mod Name;Mod Goal;Add Reqt;Del Reqt;Add Scope;Del Scope;Mod Fn;Add
Label;Mod Label;Add Physical;Del Physical']),
Win = 'Partial Match Window',
wgcreate(Win,40,00,440,570,70,0,0,1,0),
gviewer(Win,off),
wfront(Win),
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
continue_search(textbox('Chicago',12,0,4,0,32,32,1,'Cont- inue Search')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem'))],1),
clear_acpvariables,
add_header(Win),
banner(add_picture1(Win,Acp),['Please wait while Ira creates this window'],150,110),
banner(add_picture2(Win,Acp),['Please wait while Ira creates this window'],150,110),
banner(add_picture3(Win,Acp),['Please wait while Ira creates this window'],150,110),
banner(add_picture4(Win,Acp),['Please wait while Ira creates this window'],150,110),beep(60).

/* Upon completing the window several functions are needed:
   - kill the window,
   - kill the data input menus,
   - obtain partially-matched acp, get father,
     and restart search mechanism. */

continue_search(double,'Partial Match Window') :-
wkill('Partial Match Window'),
kill_menu('Objects'),
kill_menu('Other Inputs'),
get_prop(part,match,Acp),
father(Father,Acp),
matching_acps(Father).
```

```
/* This program describes the dialogue provided to elicit additional
   knowledge from the analyst when two or more partial matches are
   identified by the search mechanism. The dialogue is based around a
   window which permits access to menus for data input, and explanations
   in the window to identify which knowledge should be input by the
   analyst. */

partmatch2_dialogue(Acplist) :-
del_prop(part,matches),
set_prop(part,matches,Acplist),
install_menu('Objects',['Add Object;Mod Object;Del Object;Change Properties;Add Structure;Del
Structure;Add Movement;Del Movement']),
install_menu('Other Inputs',['Mod Name;Mod Goal;Add Reqt;Del Reqt;Add Scope;Del Scope;Mod Fn;Add
Label;Mod Label;Add Physical;Del Physical']),
Win = 'Partial Matches Window',
wgcreate(Win,40,00,440,570,70,0,0,1,0),
gviewer(Win,off),
wfront(Win),
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
continue_searches(textbox('Chicago',12,0,4,0,32,32,1,'Cont- inue Search')),
see_target(textbox('Chicago',12,0,4,0,32,32,1,'See Target Problem'))],1),
clear_acpvariables,
add_header(Win),
banner(add_picture5(Win,Acplist),['Please wait while Ira creates this window'],150,110),
banner(add_picture6(Win,Acplist),['Please wait while Ira creates this window'],150,110),
banner(add_picture7(Win,Acplist),['Please wait while Ira creates this window'],150,110),
banner(add_picture8(Win,Acplist),['Please wait while Ira creates this window'],150,110),beep(60).

/* Upon completing the window several functions are needed:
   - kill the window,
   - kill the data input menus,
   - obtain partial list, get father, and restart search mechanism. */

continue_searches(double,Win) :-
wkill(Win),
kill_menu('Objects'),
kill_menu('Other Inputs'),
get_prop(part,matches,Acplist),
Acplist = [First_acp|Rest],
father(Father,First_acp),
matching_acps(Father).
```

```
/* The first programs manages the end of search, to send the process
   to a successful or unsuccessful match - if, when searching is stopped,
   ACP = 'top' then match has failed, otherwise matching succeeded. */

stop_searching(Resulting_acp) :-
Resulting_acp = 'top',
unsuccessful_match,!.

stop_searching(Resulting_acp) :-
successful_match(Resulting_acp).

/* The program for a successful match works in two ways. A mdialogue is
   presented to state successful match, then the relevant explanation
   window is retrieved to describe the analogy. This is as far as the tool
   goes. */

successful_match(Selected_acp) :-
beep(60),mdialog(85,100,250,350,
[button(210,127,20,100,'Continue'),
text(20,20,96,310,'Ira has successfully retrieved a candidate abstraction for your new problem. This
abstraction can be retrieved by pressing CONTINUE. You should consider this abstraction to decide whether
it sufficiently represents your new computer system.'),
text(120,20,64,310,'If the abstraction is appropriate Ira will retrieve candidate reusable specifications with
which to develop a specification for the new problem.')],Btn),
del_prop(reset,dialogue),
del_prop(reset,list),
set_prop(reset,dialogue,finalgood),
set_prop(reset,list,Selected_acp),
fetch_explanation(Selected_acp).

/* The window also contains the relevant dialogue for a failed match.
   When the analogy engine fails it invites the analyst to input more data
   about the new problem */

unsuccessful_match :-
beep(60),mdialog(85,100,150,350,
[button(120,127,20,100,'Continue'),
text(20,20,100,310,'Ira was unable to retrieve any candidate abstractions for the new problem. Input more
data about the new problem then search again. To do this CONTINUE then use pull down menus to input
more data about the problem.')],Btn),!.
```

```
/* Window to explain the OAP structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_oap('Explain Object Allocation Problem') :-
wgcreate('Explain Object Allocation Problem', 40,0,440,570,70,0,0,1,0),
explain_oap('Explain Object Allocation Problem'),
gviewer('Explain Object Allocation Problem',off),
wfront('Explain Object Allocation Problem').

explain_oap(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
oap_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
oap_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
add_pic(Win,exp_oap,[
textbox('Times',14,4,5,5,20,280,1,'The Object Allocation Problem'),
textbox('Times',12,0,30,10,60,260,0,'The Object Allocation Problem represents a type of domain which
involves allocating objects to an allocation providing that they meet given constraints. Objects are only moved
to the allocation if they have properties which meet the necessary constraints.'),
textbox('Times',12,0,96,10,36,260,0,'In general this domain involves moving many objects to an allocation
rather than only moving a single object.'),
textbox('Times',12,0,138,10,60,260,0,'The movement of objects to the allocation is controlled by the
information system, so the system must allocate objects. The information system must check the properties of
objects before attempting to allocate them.'),
textbox('Times',12,0,150,285,24,195,0,'In the diagram below many objects move from Space1 to the
Allocation.'),
]),
mappings_list(Win),
check_seetarget(Win),
res_open('exploap'),
add_pic(Win,exploap,picture(250,10,155,260,resource(exploap,exploap))).

oap_return(double,Win) :-
wkill(Win),
reset_dialogue.

oap_help(double,Win) :-
oap_helpwindow(Window).

oap_helpwindow('OAP Help Window') :-
wgcreate('OAP Help Window', 40,0,440,570,70,0,0,1,0),
help_oap('OAP Help Window'),
gviewer('OAP Help Window',off),
wfront('OAP Help Window').

help_oap(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
oap_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,exp_oap,[
textbox('Times',14,4,5,5,20,280,1,'The Object Allocation Problem'),
textbox('Times',12,0,30,10,60,400,0,'There is no additional help for the Object Allocation Problem.')]).

oaphelp_return(double,Win) :-
wkill('OAP Help Window').
```

```
/* Window to explain the OAP structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_oapaa('Explain Constrained Object Allocation Problem') :-
wgcreate('Explain Constrained Object Allocation Problem', 40,0,440,570,70,0,0,1,0),
explain_oapaa('Explain Constrained Object Allocation Problem'),
gviewer('Explain Constrained Object Allocation Problem',off),
wfront('Explain Constrained Object Allocation Problem').

explain_oapaa(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
oapaa_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
oapaa_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
add_pic(Win,exp_oapaa,[
textbox('Times',14,4,5,5,20,280,1,'The Constrained Object Allocation Problem'),
textbox('Times',12,0,30,10,36,260,0,'The Constrained Object Allocation Problem represents a type of
domain in which objects are allocated to slots if they meet given constraints.'),

textbox('Times',12,0,72,10,60,260,0,'Requirements are fulfilled by resources if each requirement and
resource share the same constraints, so this problem is a matching problem. Allocation of requirements to
resources is maximised in several possible ways:'),
textline('Times',12,0,138,10,'* Initially sorting objects and slots by their priority'),
textline('Times',12,0,150,10,'* A two-pass matching process,'),
textline('Times',12,0,162,10,'* Linear Programming Techniques.'),
textbox('Times',12,0,180,10,36,260,0,'This problem type is similar to the childrens test in which the child
must place the correctly-shaped peg in  the same-shaped slot.'),
textbox('Times',12,0,150,285,60,195,0,'In the diagram below Requirements are allocated to an Allocation
containing many Resources. Both Requirements and Resources have different properties or constraints.')]),
mappings_list(Win),
check_seetarget(Win),
res_open('exploapaa'),
add_pic(Win,exploapaa,picture(250,10,155,250,resource(exploapaa,
exploapaa))).

oapaa_return(double,Win) :-
wkill(Win),
reset_dialogue.

oapaa_help(double,Win) :-
oapaa_helpwindow(Window).

oapaa_helpwindow('OAPAA Help Window') :-
wgcreate('OAPAA Help Window', 40,0,440,570,70,0,0,1,0),
help_oapaa('OAPAA Help Window'),
gviewer('OAPAA Help Window',off),
wfront('OAPAA Help Window').

help_oapaa(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
oapaahelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,hoapaa,[
textbox('Times',14,4,5,5,20,280,1,'The Constrained Object Allocation Problem'),
textbox('Times',12,0,30,10,36,400,0,'One example of the Object Allocation Problem is a video hiring
function in which videos are hired to hotels on a monthly basis. However, allocation only occurs if hotel
requirements match video details, i.e. both share the same constraints.'),
```

```
textbox('Times',12,0,72,10,24,400,0,'The allocation system is run at the beginning of each month. It has a
list of all hotel requirements to be met and video copies (resources) with which to filfil them.'),
textbox('Times',12,0,102,10,48,400,0,'The system compares each video copy to a hotel requirement and
allocates the copy if it meets the hotels requirements. This process can be refined by priority scheduling and
sorting. Constraints applicable to both video copies and hotel needs include:'),
textline('Times',12,0,156,10,'* language, e.g. french and english language films only,'),
textline('Times',12,0,168,10,'* censorship rating, e.g. no X-rated films.'),
textline('Times',12,0,186,10,'The following mappings exist:'),
textline('Times',12,0,204,10,'* Hotel Requirements  map to  Requirements,'),
textline('Times',12,0,216,10,'* Video Copies  map to  Resources.')]),
res_open('egoapaa'),
add_pic(Win,egoapaa,picture(240,10,175,300,resource(egoapaa,
egoapaa))).

oapaahelp_return(double,Win) :-
wkill('OAPAA Help Window').
```

```
findall(Data,get_physical(Data),Datalist),
Datalist = [First|Rest],
mdialog(58,125,230,300,
[button(200,20,20,160,'Remove Attribute'),
button(200,220,20,60,'Cancel'),
text(10,10,64,280,'Select the requirement which you wish to undo, then click REMOVE ATTRIBUTE:'),
menu(80,50,98,200,Datalist,[First],List)],Btn,
check_delphy(List)),
List = [L|Rest],
find_physical(Object,Attribute,L),
retract(target_phyprop(Object,Attribute)).

check_delphy(D,B,List) :-
length(List,Total),Total=\=1,
beep(60),message(['You must select one attribute from the menu']),!,fail.
check_delphy(D,B,_) :- !.

/* Specialised version of the string-matching menu eliciter, to read the
   correct selection from the menu. This program is simple, since there
   are basically only two objects to concatenate. */

get_physical(T2) :-
target_phyprop(O,A),
concat(',',A,T1),
concat(O,T1,T2).

find_physical(O,A,Selected) :-
target_phyprop(O,A),
concat(',',A,T1),
concat(O,T1,T2),
compare(=,T2,Selected).

/* Deletion of a condition program - it is constructed differently from the
   program to input conditions. An additional control is necessary to say
   when no conditions are available to be deleted. */

delete_condition :-
not target_cdata(_,_),
mdialog(100,100,100,300,
[button(70,100,20,100,'Continue'),
text(10,10,48,280,'There are no conditions to be deleted. Please click CONTINUE to return to the
menu.')],Btn),!.

delete_condition :-
fetch_condition(F,C),
target_ddata(F,O,_,_,_),
mdialog(100,100,170,300,
[button(80,210,20,60,'Next'),
button(140,210,20,60,'Cancel'),
button(110,210,20,60,'Delete'),
text(10,10,48,280,'Please use the next button to select the condition which you wish to remove, then click
DELETE:'),
text(80,10,16,100,F),
text(100,10,16,100,O),
text(120,10,16,40,'when'),
text(140,10,16,190,C)],Btn,delcheck(F,C)),
delete_condition.

/* We shall attempt to do the deletions during the button selections. */
```

```
/* Window to explain the OCP structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_ocp('Explain Object Containment Problem') :-
wgcreate('Explain Object Containment Problem',40,0,440,570,70,0,0,1,0),
explain_ocp('Explain Object Containment Problem'),
gviewer('Explain Object Containment Problem',off),
wfront('Explain Object Containment Problem').

explain_ocp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
ocp_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
add_pic(Win,exp_ocp,[
textbox('Times',14,4,5,5,20,280,1,'The Object Containment Problem (OCP)'),
textbox('Times',12,0,30,10,36,260,0,'The OCP represents a general type of problem in which objects leave
a slot which acts as a store for these objects.'),
textbox('Times',12,0,72,10,48,260,0,'In the diagram below objects are held in a slot which acts as a store
then move from the slot to an area outside the store, represented here as Space2. There are no conditions
which control movement of objects from the store.'),
textbox('Times',12,0,126,10,36,260,0,'The OCP can represent a wide range of known problems and
information systems, including stock control and library problems.'),
textbox('Times',12,0,168,10,36,260,0,'In the diagram below Objects move from a container called to a Slot
to a space outside the Slot called Space2.'),
]),
mappings_list(Win),
check_seetarget(Win),
res_open('explocp'),
add_pic(Win,explocp,
picture(250,10,165,300,resource(explocp,explocp))).

ocp_return(double,Win) :-
wkill(Win),
reset_dialogue.

ocp_help(double,Win) :-
ocp_helpwindow(Window).

ocp_helpwindow('OCP Help Window') :-
wgcreate('OCP Help Window', 40,0,440,570,70,0,0,1,0),
help_ocp('OCP Help Window'),
gviewer('OCP Help Window',off),
wfront('OCP Help Window').

help_ocp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocphelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,hocp,[
textbox('Times',14,4,5,85,20,280,1,'The Object Containment Problem (OCP)'),
textline('Times',12,0,30,60,'There is no additional help to describe the Object Containment Problem.')
]).

ocphelp_return(double,Win) :-
wkill('OCP Help Window').
```

```
/* Window to explain the OCP-AA structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_ocpaa('Explain Non-renewable Resource Mgmt Problem') :-
wgcreate('Explain Non-renewable Resource Mgmt Problem', 40,0,440,570,70,0,0,1,0),
explain_ocpaa('Explain Non-renewable Resource Mgmt Problem'),
gviewer('Explain Non-renewable Resource Mgmt Problem',off),
wfront('Explain Non-renewable Resource Mgmt Problem').

explain_ocpaa(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpaa_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
ocpaa_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
add_pic(Win,exp_ocpaa,[
textbox('Times',14,4,5,5,32,280,1,'The Non-renewable Resource Management Problem (RMP)'),
textbox('Times',12,0,45,10,60,260,0,'The non-renewable RMP represents problems which maintain a store
of objects. Objects are held in a slot which`acts as a store and leave the slot to go into the world. They are
replenished by objects from a different source.'),
textbox('Times',12,0,111,10,24,260,0,'Objects which leave the slot are beyond the control of the associated
information system.'),
textbox('Times',12,0,141,10,60,260,0,'When the number of objects in the slot reaches a level (often a
minimum quantity of objects) the information system initiates a movement of objects from the world to the
slot This replenishment always ensures that the slot has sufficient objects.'),
textbox('Times',12,0,207,10,60,260,0,'The requirement of the information system is to ensure that the store
always contains a minimum quantity of objects.'),
textbox('Times',12,0,146,290,36,195,0,'Information system functions for this problem type include
Receive, Dispatch and Accept.'),
textbox('Times',12,0,195,290,60,195,0,'In this diagram the world is represented as a space. Objects move
into the slot (the store) from Space1 and move out of the slot into Space2.'),
]),
mappings_list(Win),
check_seetarget(Win),
res_open('explocpaa'),
add_pic(Win,explocpaa,
picture(250,10,165,400,resource(explocpaa,explocpaa))).

ocpaa_return(double,Win) :-
wkill(Win),
reset_dialogue.

ocpaa_help(double,Win) :-
ocpaa_helpwindow(Window).

ocpaa_helpwindow('Non-renewable Resource Help Window') :-
wgcreate('Non-renewable Resource Help Window', 40,0,440,570,70,0,0,1,0),
help_ocpaa('Non-renewable Resource Help Window'),
gviewer('Non-renewable Resource Help Window',off),
wfront('Non-renewable Resource Help Window').

help_ocpaa(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpaahelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,hocpaa,[
textbox('Times',14,4,5,5,32,280,1,'The Non-renewable Resource Management Problem (RMP)'),
textbox('Times',12,0,45,10,36,360,0,'The non-renewable RMP is typical of types of stock control
```

```
problem, including warehousing and hospital supplies maintenance. An example hospital supplies problem is
described below.'),
textbox('Times',12,0,88,10,36,360,0,'A Hospital Cancer ward has a supply of medicines and drugs used
for treating patients in the ward. The level of these drugs is monitored by a computer system which informs
staff when levels are low.'),
textbox('Times',12,0,130,10,48,360,0,'Staff request new drug consignments from suppliers of the drug
when indicated to do so by the information system. Suppliers then send the consignment to restock the wards
drug supplies. The Ward domain is represented graphically below.'),
textline('Times',12,0,184,10,'In the example the following mappings exist:'),
textline('Times',12,0,196,10,'* Drugs/Medicines  map to  Objects,'),
textline('Times',12,0,208,10,'* Ward Supply  maps to  Store,'),
textline('Times',12,0,220,10,'* Supplier  maps to  Space1,'),
textline('Times',12,0,232,10,'* Patients  maps to  Space2.'),
]),
res_open('egocpaa'),
add_pic(Win,egocpaa,picture(250,10,170,440,resource(egocpaa,
egocpaa))).

ocpaahelp_return(double,Win) :-
wkill(Win).
```

```
/* Window to explain the OCP-AB structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms. */

explanation_ocpab('Explain Renewable Resource Mgmt Problem') :-
wgcreate('Explain Renewable Resource Mgmt Problem', 40,0,440,570,70,0,0,1,0),
explain_ocpab('Explain Renewable Resource Mgmt Problem'),
gviewer('Explain Renewable Resource Mgmt Problem',off),
wfront('Explain Renewable Resource Mgmt Problem').

explain_ocpab(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpab_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
ocpab_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
add_pic(Win,exp_ocpab,[
textbox('Times',14,4,5,5,32,280,1,'The Renewable Resource Management Problem (RRMP)'),
textbox('Times',12,0,45,10,48,260,0,'The RRMP represents problems involved in maintaining a store of
objects which are taken from and returned to the store or slot. These problems are often referred to as Library
problems.'),
textbox('Times',12,0,99,10,48,260,0,'The movement of objects to and from the slot is not initiated by the
information system. Rather this system monitors and records the whereabouts of objects outside the slot.'),
textbox('Times',12,0,153,10,36,260,0,'The return of objects from the outside world to the slot is often
controlled by a time-limit or date by which the return must be made.'),
textbox('Times',12,0,195,10,36,260,0,'Information system functions for this problem type include lend,
borrow and return.'),
textbox('Times',12,0,153,290,60,200,0,'In the diagram below the world is represented as a space, and
objects move into and out of the Space2.')]),
mappings_list(Win),
check_seetarget(Win),
res_open('explocpab'),
add_pic(Win,explocpab,
picture(250,10,165,300,resource(explocpab,explocpab))).

ocpab_return(double,Win) :-
wkill(Win),
reset_dialogue.

ocpab_help(double,Win) :-
ocpab_helpwindow(Window).

ocpab_helpwindow('OCP Help Window') :-
wgcreate('Renewable Resource Help Window', 40,0,440,570,70,0,0,1,0),
help_ocpab('Renewable Resource Help Window'),
gviewer('Renewable Resource Help Window',off),
wfront('Renewable Resource Help Window').

help_ocpab(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpabhelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,hocpab,[
textbox('Times',14,4,5,5,32,280,1,'The Renewable Resource Management Problem (RRMP)'),
textbox('Times',12,0,45,10,36,350,0,'The RRMP represents most types of library or hiring problems. The
example below describes a company which hires heavy-duty machinery to builders.'),
textbox('Times',12,0,87,10,72,350,0,'The tool-hiring company has a store of machines which are lent to
building companies and private individuals. A machine is lent for a specified length of time then returned to
the company. The system monitors these loans and checks for overdue loans.'),
```

```
textline('Times',12,0,141,10,'The following mappings exist:'),
textline('Times',12,0,159,10,'* Tool  maps to  Object,'),
textline('Times',12,0,171,10,'* Company  maps to  Slot,'),
textline('Times',12,0,183,10,'* Builder  maps to  Space2.')]),
res_open('egocpab'),
add_pic(Win,egocpab,picture(230,10,175,370,resource(egocpab,
egocpab))).

ocpabhelp_return(double,Win) :-
wkill(Win).
```

```
/* Window to explain the OCP-BA structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_ocpba('Explain Structured Resource Mgmt Problem') :-
wgcreate('Explain Structured Resource Mgmt Problem',40,0,440,570,70,0,0,1,0),
explain_ocpba('Explain Structured Resource Mgmt Problem'),
gviewer('Explain Structured Resource Mgmt Problem',off),
wfront('Explain Structured Resource Mgmt Problem').

explain_ocpba(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpba_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
ocpba_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
phymatch_ocpba(Win),
add_pic(Win,exp_ocpba,[
textbox('Times',14,4,5,5,32,280,1,'The Structured Non-renewable Resource Management Problem
(RMP)'),
textbox('Times',12,0,45,10,48,260,0,'The non-renewable RMP represents problems involved in
maintaining a store of objects. This store is divided into many small slots, each of which contains objects.'),
textbox('Times',12,0,99,10,48,260,0,'Many objects leave each small slot to go into the world and are
replenished by objects from a different source. Objects which leave the small slot are beyond the control of
the associated information system.'),
textbox('Times',12,0,153,10,60,260,0,'When the number of objects in any small slot reaches a level (often
a minimum quantity of objects) the system initiates a movement of objects from the world to that small slot.
This replenishment ensures that small slots have sufficient objects.'),
textbox('Times',12,0,219,10,60,260,0,'The requirement of the information system is to ensure that each
small slot always contains a minimum quantity of objects.'),
textbox('Times',12,0,146,290,36,200,0,'Information system functions for this problem type include
Receive, Dispatch and Accept.'),
textbox('Times',12,0,188,290,60,195,0,'In this diagram the world is represented as a Space. Objects move
into the Smallslot via the Slot from Space1 and move out of the Smallslot via the Slot into Space2.'),
]),
mappings_list(Win),
check_seetarget(Win),
res_open('explocpba'),
add_pic(Win,explocpba,
picture(250,10,170,400,resource(explocpba,explocpba))).

ocpba_return(double,Win) :-
wkill(Win),
reset_dialogue.

ocpba_help(double,Win) :-
ocpba_helpwindow(Window).

ocpba_helpwindow('Structured Non-renewable Resource Problem Help Window') :-
wgcreate('Structured Non-renewable Resource Problem Help Window', 40,0,440,570,70,0,0,1,0),
help_ocpba('Structured Non-renewable Resource Problem Help Window'),
gviewer('Structured Non-renewable Resource Problem Help Window',off),
wfront('Structured Non-renewable Resource Problem Help Window').

help_ocpba(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpbahelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,hocpba,[
```

```
textbox('Times',14,4,5,5,32,280,1,'The Structured Non-renewable Resource Management Problem
(RMP)'),
textbox('Times',12,0,45,10,36,350,0,'The non-renewable RMP represents most types of complex stock
control problems. The following example describes one instance of this stock control problem: maintaining a
stock of office stationary.'),
textbox('Times',12,0,87,10,36,350,0,'A large organisation uses an information system to control use of its
stationary. When levels of each item (e.g. biros) reach a given limit a new quantity of that item is ordered
from the relevant wholesalers.'),
textbox('Times',12,0,129,10,48,350,0,'Staff in the organisation use stationary from the cupboards are
necessary, and once a week the stationary is checked to identify current levels of each item. The information
system then decides upon the need for new stationary and prints supplier orders.'),
textline('Times',12,0,183,10,'The following mappings exist:'),
textline('Times',12,0,195,10,'* Stationary  maps to  Objects,'),
textline('Times',12,0,207,10,'* Slot  maps to  Organisation,'),
textline('Times',12,0,219,10,'* Smallslot  maps to  Container of each Stationary Type,'),
textline('Times',12,0,231,10,'* Space1  maps to  Stationary Suppliers,'),
textline('Times',12,0,243,10,'* Space2  maps to  Employees.')]),
res_open('egocpba'),
add_pic(Win,egocpba,picture(253,10,175,370,resource(egocpba,
egocpba))).

ocpbahelp_return(double,Win) :-
wkill(Win).

/* Routines for physical match option. */

phymatch_ocpba(Win) :-
physical_acp(ocpba),
add_tools(Win,[
physical_ocpba(textbox('Chicago',12,0,6,0,32,32,1,'Physical Match'))],1),!.

phymatch_ocpba(Win) :- !.

physical_ocpba(double,Win) :-
fetch_explanation(pocpba).
```

```
/* Window to explain the OCP-AA structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_ocpbb('Object Recording Problem') :-
wgcreate('Object Recording Problem', 40,0,440,570,70,0,0,1,0),
explain_ocpbb('Object Recording Problem'),
gviewer('Object Recording Problem',off),
wfront('Object Recording Problem').

explain_ocpbb(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpbb_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
ocpbb_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
add_pic(Win,exp_ocpbb,[
textbox('Times',14,4,5,5,32,280,1,'The Object Recording Problem'),
textbox('Times',12,0,45,10,48,260,0,'The Object Recording Problem represents domains which record the
existence of objects in a slot. It monitors the movements of objects in the domain and does not initiate these
object movements itself.'),
textbox('Times',12,0,99,10,48,260,0,'The information system is interested in two types of movement -
those into the slot and those out of the slot again. The system records data about each object in the slot.'),
textbox('Times',12,0,153,10,48,260,0,'Ira differentiates between the original source and final destination of
the objects. Generally objects will not return to their original location upon leaving the slot, although this may
happen.'),
textbox('Times',12,0,207,10,24,260,0,'The functions of the information system associated with this
problem type include Record and Update.'),
textbox('Times',12,0,153,285,60,195,0,'In the diagram below Objects exist in a Slot. Objects originally
enter the Slot from a source Space1 and leave it for a destination Space2.')]),
mappings_list(Win),
check_seetarget(Win),
res_open('explocpbb'),
add_pic(Win,explocpbb,
picture(250,10,165,400,resource(explocpbb,explocpbb))).

ocpbb_return(double,Win) :-
wkill(Win),
reset_dialogue.

ocpbb_help(double,Win) :-
ocpbb_helpwindow(Window).

ocpbb_helpwindow('Object Recording Help Window') :-
wgcreate('Object Recording Help Window', 40,0,440,570,70,0,0,1,0),
help_ocpbb('Object Recording Help Window'),
gviewer('Object Recording Help Window',off),
wfront('Object Recording Help Window').

help_ocpbb(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
ocpbbhelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,hocpbb,[
textbox('Times',14,4,5,5,32,280,1,'The Object Recording Problem'),
textbox('Times',12,0,45,10,48,350,0,'The Personnel System described during the problem elicitation phase
of Ira is one instance of an Object Recording Problem. This system monitors the movement of staff to and
from a company, and records data on staff in the company.'),
textbox('Times',12,0,99,10,48,350,0,'Object Recording Systems are quite simple. They only monitor the
```

```
movements of objects such as personnel. It is possible that objects such as Staff can return to the same place
from where they came (e.g. the job agency).'),
textline('Times',12,0,153,10,'The following mappings with the Personnel System exist:'),
textline('Times',12,0,165,10,'* Staff  map to  Objects,'),
textline('Times',12,0,177,10,'* Organisation  maps to  Slot,'),
textline('Times',12,0,189,10,'* Outside World  maps to  Space1.')]),
res_open('egocpbb'),
add_pic(Win,egocpbb,picture(230,10,155,450,resource(egocpbb,
egocpbb))).

ocpbbhelp_return(double,Win) :-
wkill(Win).
```

```
/* Window to explain the OMP structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_omp('Explain Object Monitoring Problem') :-
wgcreate('Explain Object Monitoring Problem',40,0,440,570,70,0,0,1,0),
explain_omp('Explain Object Monitoring Problem'),
gviewer('Explain Object Monitoring Problem',off),
wfront('Explain Object Monitoring Problem').

explain_omp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
omp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
omp_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
phymatch_omp(Win),
add_pic(Win,exp_omp,[
textbox('Times',14,4,5,5,20,280,1,'The Object Monitoring Problem'),
textbox('Times',12,0,30,10,60,260,0,'The Object Monitoring Problem represents domains in which many
objects move in a space and risk collision. The purpose of the information system is to monitor object
movement and provide imminent warning of potential collisions.'),
textbox('Times',12,0,96,10,60,260,0,'Each object in the domain is surrounded by a space or slot which
protects the object from collision. No other object may enter this space else the space controller is warned and
makes the neccesary actions to avoid the collision.'),
textbox('Times',12,0,162,10,48,260,0,'When a slot contains more than two objects controllers must act to
remove additional objects from each slot. This is done by issuing commands which affect the movement of
objects.'),
textbox('Times',12,0,216,10,36,260,0,'The information system is limited to monitoring and controlling the
domain. Object movement is beyond control of the system.'),
textbox('Times',12,0,162,285,84,195,0,'In the diagram below many objects move freely in and between
slots. Each slot may contain none, one or many objects, although the system should warn controllers when
many objects are in the same slot.')
]),
mappings_list(Win),
check_seetarget(Win),
res_open('explomp'),
add_pic(Win,explomp,
picture(250,10,160,300,resource(explomp,explomp))).

omp_return(double,Win) :-
wkill(Win),
reset_dialogue.

omp_help(double,Win) :-
omp_helpwindow(Window).

omp_helpwindow('Object Monitoring Help Window') :-
wgcreate('Object Monitoring Help Window', 40,0,440,570,70,0,0,1,0),
help_omp('Object Monitoring Help Window'),
gviewer('Object Monitoring Help Window',off),
wfront('Object Monitoring Help Window').

help_omp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
omphelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,homp,[
textbox('Times',14,4,5,105,20,200,0,'The Object Monitoring Problem'),
```

```
textbox('Times',12,0,30,10,36,400,0,'The Object Monitoring Problem can be extended to monitor for plan
adherence as well as collision detection. Objects often move according to a predetermined plan, and many
instances of this problem type incorporate plan adherence monitoring.'),
textbox('Times',12,0,72,10,24,400,0,'The following example can be extended to incorporate plan
adherence as well as collision detection monitoring.'),
textbox('Times',12,0,102,10,48,400,0,'An underground railway is broken down into a number of track
sections to ensure passenger safety. Each track section may only contain one train, otherwise the signalman
keeps trains apart by changing relevant signals and informing drivers.'),
textbox('Times',12,0,156,10,24,400,0,'The information system can also monitor the direction of trains to
ensure they are travelling in the right direction according to the railway timetable. Mappings are:'),
textline('Times',12,0,186,10,'* Train  maps to  Object'),
textline('Times',12,0,198,10,'* Space  maps to  Track Section.')]),
res_open('egomp'),
add_pic(Win,egomp,picture(230,10,175,420,resource(egomp,egomp))).

omphelp_return(double,Win) :-
wkill(Win).

/* Routines for physical match option. */

phymatch_omp(Win) :-
physical_acp(omp),
add_tools(Win,[
physical_omp(textbox('Chicago',12,0,6,0,32,32,1,'Physical Match'))],1),!.

phymatch_omp(Win) :- !.

physical_omp(double,Win) :-
fetch_explanation(pomp).
```

```
/* Window to explain the OCP-AA structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_opp('Explain Simple Spatial Object Problem') :-
wgcreate('Explain Simple Spatial Object Problem', 40,0,440,570,70,0,0,1,0),
explain_opp('Explain Simple Spatial Object Problem'),
gviewer('Explain Simple Spatial Object Problem',off),
wfront('Explain Simple Spatial Object Problem').

explain_opp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
opp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
opp_help(textbox('Chicago',12,0,6,0,32,32,1,'More Help'))],1),
phymatch_opp(Win),
add_pic(Win,exp_opp,[
textbox('Times',14,4,5,5,20,280,1,'The Object Positioning Problem'),
textbox('Times',12,0,30,10,48,260,0,'The Object Positioning Problem monitors the position of objects with
regard to a specific space or slot. The aim of the information system is to ensure that the slot is always
occupied by an object.'),
textbox('Times',12,0,84,10,60,260,0,'The information system monitors the movement of objects with
regard to the slot, and if the slot is unattended then the system informs the controller who takes appropriate
action to reoccupy the slot with another object.'),
textbox('Times',12,0,150,10,48,260,0,'It is possible that the object which vacated the slot may also be
directed by the controller to reoccupy the slot.'),
textbox('Times',12,0,180,10,24,260,0,'The information system is restricted to monitoring the domain and
advising a controller when necessary.'),
textbox('Times',12,0,150,285,60,195,0,'In the diagram below the objects must be positioned in a Slot. The
Object leaves the Slot to Space2, then a new Object enters the Slot from Space1.')]),
mappings_list(Win),
check_seetarget(Win),
res_open('explopp'),
add_pic(Win,explopp,
picture(250,10,165,300,resource(explopp,explopp))).

opp_return(double,Win) :-
wkill(Win),
reset_dialogue.

opp_help(double,Win) :-
opp_helpwindow(Window).

opp_helpwindow('Simple Spatial Object Help Window') :-
wgcreate('Simple Spatial Object Help Window', 40,0,440,570,70,0,0,1,0),
help_opp('Simple Spatial Object Help Window'),
gviewer('Simple Spatial Object Help Window',off),
wfront('Simple Spatial Object Help Window').

help_opp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
opphelp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,hopp,[
textbox('Times',14,4,5,5,20,280,1,'The Object Positioning Problem'),
textbox('Times',12,0,30,10,36,350,0,'An example of an Object Positioning Problem is Coastguard patrols.
Patrol boats monitor coastal waters to ensure that no small craft pass by unnoticed.'),
textbox('Times',12,0,72,10,24,350,0,'The information system checks the position of boats to ensure they
```

```
maintain an effective cordon through which no boats may pass illegally.'),
textbox('Times',12,0,102,10,48,350,0,'If radar suggests that a boat is found to be out of position then the system informs the controller who advises other boats to fill the space vacated by the original boat. The same boat may be redirected back to the same patrol zone.'),
textline('Times',12,0,156,10,'In this example the following mappings occured:'),
textline('Times',12,0,174,10,'*  Object  maps to  Coastguard Boat,'),
textline('Times',12,0,186,10,'*  Slot  maps to  Patrol Zone.')]),
res_open('egopp'),
add_pic(Win,egopp,picture(230,10,175,370,resource(egopp,egopp))).

opphelp_return(double,Win) :-
wkill(Win).

/* Routines for physical match option. */

phymatch_opp(Win) :-
physical_acp(opp),
add_tools(Win,[
physical_opp(textbox('Chicago',12,0,6,0,32,32,1,'Physical  Match'))],1),!.

phymatch_opp(Win) :- !.

physical_opp(double,Win) :-
fetch_explanation(popp).
```

```
/* Window to explain the OCP-BA structure to the analyst - it is a standard
   window accessed by all explanation\retrieval mechanisms */

explanation_pocpba('Explain Warehousing Problem') :-
wgcreate('Explain Warehousing Problem',40,0,440,570,70,0,0,1,0),
explain_pocpba('Explain Warehousing Problem'),
gviewer('Explain Warehousing Problem',off),
wfront('Explain Warehousing Problem').

explain_pocpba(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
pocpba_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,exp_pocpba,[
textbox('Times',14,4,5,5,32,280,1,'The Warehousing Problem'),
textbox('Times',12,0,33,10,72,260,0,'The warehousing problem is a typical stock control problem. A
warehouse contains stock which is used by an organisation for sales or manufacturing. Stock is held in many
bins which are replenished from incoming supplies when these stocks begin to run low.'),
textbox('Times',12,0,111,10,60,260,0,'Stock enters the warehouse through the good-in where it is
normally checked. It leaves the warehouse to the sales, delivery or manufacturing departments. The
information system monitors levels of stock in the bins to warn of low stock levels.'),
textline('Times',12,0,147,285,'The following mappings exist:'),
textline('Times',12,0,159,285,'* Stock  map to  Objects,'),
textline('Times',12,0,171,285,'* Warehouse  maps to  Store,'),
textline('Times',12,0,183,285,'* Stock Bin  maps to  Smallslot,'),
textline('Times',12,0,195,285,'* Supplier  maps to  Space1,'),
textline('Times',12,0,207,285,'* Goods-out  maps to  Space2.')]),
mappings_list(Win),
res_open('explpocpba'),
add_pic(Win,explpocpba,
picture(220,10,200,400,resource(explpocpba,explpocpba))).

pocpba_return(double,Win) :-
wkill(Win),
reset_dialogue.
```

```
/* Window to explain transport instantiation of the OMP domain. */

explanation_pomp('Explain Network Transport Collision Problem') :-
wgcreate('Explain Network Transport Collision Problem',
40,0,440,570,70,0,0,1,0),
explain_pomp('Explain Network Transport Collision Problem'),
gviewer('Explain Network Transport Collision Problem',off),
wfront('Explain Network Transport Collision Problem').

explain_pomp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
pomp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,exp_pomp,[
textbox('Times',14,4,5,5,20,280,1,'The Network Transport Collision Problem'),
textbox('Times',12,0,30,10,60,260,0,'The Network Transport Collision Problem represents many transport
domains in which vehicles, such as ships, aircraft and trains, may collide. The aim of the information system
is to monitor vehicle movement and warn of potential collisions.'),
textbox('Times',12,0,96,10,48,260,0,'Vehicle movement is constrained by a network of unidirectional lanes
along which vehicles should move. Lanes may by two-dimensional (Railways) or three-dimensional
(Airways).'),
textbox('Times',12,0,150,10,48,260,0,'In either case vehicles are still protected by a space which limits the
number of vehicles in the space. A controller warns vehicle operators in the space is violated.'),
textline('Times',12,0,150,285,'The following mappings exist:'),
textline('Times',12,0,162,285,'*'),
textline('Times',12,0,174,285,'*'),
textline('Times',12,0,186,285,'*'),
textline('Times',12,0,162,295,'Vehicle  maps to  Object,'),
textline('Times',12,0,174,295,'Space  maps to  Safety Zone,'),
textline('Times',12,0,186,295,'Path  maps to  Airlane/Railway Track.')]),
mappings_list(Win),
res_open('explpomp'),
add_pic(Win,explpomp,
picture(200,10,220,380,resource(explpomp,explpomp))).

pomp_return(double,Win) :-
wkill(Win),
reset_dialogue.
```

```
/* Window to explain the physical instantiation of the OCP-BB abstraction.
   */

explanation_popp('Explain Transport Positioning Problem') :-
wgcreate('Explain Transport Positioning Problem', 40,0,440,570,70,
0,0,1,0),
explain_popp('Explain Transport Positioning Problem'),
gviewer('Explain Transport Positioning Problem',off),
wfront('Explain Transport Positioning Problem').

explain_popp(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
popp_return(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1),
add_pic(Win,exp_popp,[
textbox('Times',14,4,5,5,20,280,1,'The Transport Positioning Problem'),
textbox('Times',12,0,30,10,60,260,0,'A Transport Positioning System monitors the position of manned
vehicles with regard to physical spaces. The aim of the system is to ensure that these spaces are always
occupied by at least one vehicle, such as a boat or plane.'),
textbox('Times',12,0,96,10,60,260,0,'The physical spaces occupied by vehicles are often adjacent, so
vehicles can be instructed when necessary to move from neighbouring spaces to the vacant space. As such
the vehicles attempt to form a protective net or barrier.'),
textbox('Times',12,0,162,10,24,260,0,'A vehicle may be instructed to return to the space which it
vacated.'),
textline('Times',12,0,162,285,'The following mappings were identified:'),
textline('Times',12,0,180,285,'Vehicle  maps to  Object,'),
textline('Times',12,0,192,285,'Space  maps to  Slot.')]),
mappings_list(Win),
res_open('explpopp'),
add_pic(Win,explpopp,
picture(220,10,200,400,resource(explpopp,explpopp))).

popp_return(double,Win) :-
wkill(Win),
reset_dialogue.
```

```
/* This program is simple but important - it controls the management of
   windows (i.e. selection etc) from various calls throughout Ira. The main
   program is called 'fetch_explanation'. There are two groups of the
   fetch program - first retrieve the basic ACPs, while the second
   retrieves the physical explanations for the ACPs. */

/* Basic ACP fetches */

fetch_explanation(ocp) :-
disable_menu('Control'),
explanation_ocp('Explain Object Containment Problem').

fetch_explanation(ocpaa) :-
disable_menu('Control'),
explanation_ocpaa('Explain Non-renewable Resource Mgmt Problem').

fetch_explanation(ocpab) :-
disable_menu('Control'),
explanation_ocpab('Explain Renewable Resource Mgmt Problem').

fetch_explanation(ocpba) :-
disable_menu('Control'),
explanation_ocpba('Explain Structured Resource Mgmt Problem').

fetch_explanation(ocpbb) :-
disable_menu('Control'),
explanation_ocpbb('Object Recording Problem').

fetch_explanation(omp) :-
disable_menu('Control'),
explanation_omp('Explain Object Monitoring Problem').

fetch_explanation(oap) :-
disable_menu('Control'),
explanation_oap('Explain Object Allocation Problem').

fetch_explanation(oapaa) :-
disable_menu('Control'),
explanation_oapaa('Explain Constrained Object Allocation Problem').

fetch_explanation(opp) :-
disable_menu('Control'),
explanation_opp('Explain Simple Spatial Object Problem').

/* Physical match ACP fetches. Only three physical description windows
   are accessed - otherwise control is returned to original abstract
   windows. */

fetch_explanation(pocp) :-
disable_menu('Control'),
del_prop(reset,dialogue),
set_prop(reset,dialogue,physical),
explanation_ocp('Explain Object Containment Problem').

fetch_explanation(pocpaa) :-
disable_menu('Control'),
del_prop(reset,dialogue),
set_prop(reset,dialogue,physical),
explanation_ocpaa('Explain Non-renewable Resource Mgmt Problem').
```

```
fetch_explanation(pocpab) :-
disable_menu('Control'),
del_prop(reset,dialogue),
set_prop(reset,dialogue,physical),
explanation_ocpab('Explain Renewable Resource Mgmt Problem').

fetch_explanation(pocpba) :-
disable_menu('Control'),
del_prop(reset,dialogue),
set_prop(reset,dialogue,physical),
explanation_pocpba('Explain Warehousing Problem').

fetch_explanation(popp) :-
disable_menu('Control'),
del_prop(reset,dialogue),
set_prop(reset,dialogue,physical),
explanation_popp('Explain Transport Collision Problem').

fetch_explanation(pomp) :-
disable_menu('Control'),
del_prop(reset,dialogue),
set_prop(reset,dialogue,physical),
explanation_pomp('Explain Network Transport Collision Problem').

fetch_explanation(poap) :-
disable_menu('Control'),
del_prop(reset,dialogue),
set_prop(reset,dialogue,physical),
explanation_oap('Explain Object Allocation Problem').

/* Another small set of programs is required to control the use of
   explanations in sequential processing. It is necessary to record the
   state of a dialogue before an explanation is accessed, so that the same
   state can be returned to afterwards. This is achieved using a coded set
   of set-props and get-props. They represent i) which dialogue you were
   in, 2) a list of candidate ACPs of interest to the dialogue (i.e. candidate
   fits).

   Several dialogue options exist:
   - two good matches to be selected between (manygood),
   - one partial match to be expanded upon (onepart),
   - several partial matches to be selected between (manypart),
   - final acceptance of partial matches (acceptparts),
   - final good fit upon completion of search (finalgood),
   - explanation for user identification of analogous mappings.

   The program is called 'reset dialogue' */

reset_dialogue :-
get_prop(reset,dialogue,Dialogue),
get_prop(reset,list,Acplist),
Dialogue = manygood,
enable_menu('Control'),
goodmatches_dialogue(Acplist),!.

reset_dialogue :-
get_prop(reset,dialogue,Dialogue),
get_prop(reset,list,Acplist),
```

```
Dialogue = onepart,
enable_menu('Control'),
partmatch_dialogue(Acplist),!.

reset_dialogue :-
get_prop(reset,dialogue,Dialogue),
get_prop(reset,list,Acplist),
Dialogue = manypart,
enable_menu('Control'),
partmatches_dialogue(Acplist),!.

reset_dialogue :-
get_prop(reset,dialogue,Dialogue),
get_prop(reset,list,Acplist),
Dialogue = acceptparts,
enable_menu('Control'),
acceptmatches_dialogue(Acplist),!.

reset_dialogue :-
get_prop(reset,dialogue,Dialogue),
enable_menu('Control'),
Dialogue = finalgood,!.

reset_dialogue :-
get_prop(reset,dialogue,Dialogue),
Dialogue = physical,!.

/* The following routine is called by all explanation windows. It provides
   the 'see target' tool if the explanations are called as part of two good
   matches dialogue. All windows call one routine which adds the 'see
   target' tool if the reset-dialogue variable is correct. */

check_seetarget(Win) :-
get_prop(reset,dialogue,manygood),
add_tools(Win,[see_target(textbox('Chicago',
12,0,4,0,32,32,1,'See Target Problem'))],1),!.

check_seetarget(Win) :- !.
```

```
/* This program allows analysts to identify specific mappings with objects
   in retrieved ACPs, then rerun analogous mappings based on these fixed
   analogical features. There are three features to this dialogue:
   (i)  dialogue to select from available ACPs (maximum choice 3),
   (ii) display of explanation window for selected ACP,
   (iii) display of dialogue for inputting object mappings. */

/* Initial dialogue to select the required ACP. Three dialogues exist:
   (i)  standard dialogue when several acps already exist,
   (ii) skip the dialogue when only one ACP exists,
   (iii) dialogue when no ACPs are known, and the dialogue must fail. */

inputmapping_dialogue :-
findall(Acp,rec_acpmatch(Acp),Acplist),
obtain_acps(Acplist).

/* Dialogue when no recorded acp matches can be called for object
   matching. */

obtain_acps(Acplist) :-
Acplist = [],
mdialog(100,100,130,300,[
text(10,10,80,280,'There are currently no matched abstractions which can be retrieved. Search using the current problem description in order to obtain some abstractions.'),
button(105,100,20,100,'Continue')],Btn),!.

/* Dialogue when one recorded acp match is retrieved. The initial window
   is not necessary so control is passed immediately to the object mapping
   window. */

obtain_acps(Acplist) :-
length(Acplist,T),T=1,
Acplist = [Acp],
mapobjects_dialogue(Acp),!.

/* Dialogue necessary when two or more recorded acp matches are
   identified. */

obtain_acps(Oldlist) :-
length(Oldlist,T),T>1,
acplist_spaces(Oldlist,Acplist),
Acplist = [A1,A2,A3],
mdialog(40,85,260,400,[
text(10,10,80,380,'Ira has identified two possible types of problem for your system. Please use the following buttons to examine each of these options, then select the most appropriate option below:'),
button(230,310,20,60,'Cancel'),
button(230,30,20,60,'Select'),
radio(120,20,16,375,A1,on,Sel1),
radio(150,20,16,375,A2,off,Sel2),
radio(180,20,16,375,A3,off,Sel3)],Btn,
valid_inputmapping(Sel1,Sel2,Sel3,A1,A2,A3)).

/* Several rules are required to manage valid selection of ACPs */

valid_inputmapping(D,B,Sel1,Sel2,Sel3,_,_,_) :-
Sel1='on',Sel2='on',
beep(60), message(['You must choose one abstraction. ~MPlease try again']),!,fail.

valid_inputmapping(D,B,Sel1,Sel2,Sel3,_,_,_) :-
```

```
Sel1='on',Sel3='on',
beep(60), message(['You must choose one abstraction. ~MPlease try again']),!,fail.

valid_inputmapping(D,B,Sel1,Sel2,Sel3,_,_,_) :-
Sel3='on',Sel2='on',
beep(60), message(['You must choose one abstraction. ~MPlease try again']),!,fail.

valid_inputmapping(D,B,Sel1,Sel2,Sel3,_,_,_) :-
Sel1='on',Sel2='on',Sel3='on',
beep(60), message(['You must choose one abstraction. ~MPlease try again']),!,fail.

valid_inputmapping(D,B,Sel1,Sel2,Sel3,_,_,_) :-
Sel1='off',Sel2='off',Sel3='off',
beep(60), message(['You must choose one abstraction. ~MPlease try again']),!,fail.

valid_inputmapping(D,B,Sel1,Sel2,Sel3,_,_,A3) :-
Sel3='on',A3='',
beep(60), message(['This radio cannot be selected. ~MPlease try again']),!,fail.

valid_inputmapping(D,3,Sel1,Sel2,Sel3,A1,A2,A3) :-
identify_selectacp(Sel1,Sel2,Sel3,A1,A2,A3,Selected_acp),
mapobjects_dialogue(Selected_acp).

/* Subroutine to add spaces to ACP list if necessary. */

acplist_spaces(Oldlist,Newlist) :-
length(Oldlist,T),T=2,
Oldlist = [O1,O2],
acps(O1,N1),acps(O2,N2),
Newlist = [N1,N2,' '],!.

acplist_spaces(Oldlist,Newlist) :-
length(Oldlist,T),T=3,
Oldlist = [O1,O2,O3],
acps(O1,N1),acps(O2,N2),acps(O3,N3),
Newlist = [N1,N2,N3].

/* Subroutine to identify the label of chosen ocp */

identify_selectacp(Sel1,Sel2,Sel3,A1,A2,A3,Selected_acp) :-
Sel1 = 'on',acps(Name,A1),Selected_acp is Name,!.

identify_selectacp(Sel1,Sel2,Sel3,A1,A2,A3,Selected_acp) :-
Sel2 = 'on',acps(Name,A2),Selected_acp is Name,!.

identify_selectacp(Sel1,Sel2,Sel3,A1,A2,A3,Selected_acp) :-
Sel3 = 'on',acps(Name,A3),Selected_acp is Name.

/* Dialogue which inputs specific analogous mappings with objects
   belonging`to each ACP. The objects are retrieved through a complicated
   subroutine in order to allow for one, two three or four ACP objects &
   a variable number of object mappings with these objects.

   User-identified analogous mappings are identified by increasing the
   score of the object mapping by 1000, so all object mappings with
   scores => 1000 are prespecified by the analyst. When generating
   a fixed mapping either increase the score of an existing mapping or
   create a new mapping with score 1000. */
```

```
mapobjects_dialogue(Acp) :-
getobjects(Olist,Acp),
getmappings(Mlist,Olist,Acp),
Olist = [O1,O2,O3,O4,O5],
Mlist = [M1,M2,M3,M4,M5],
mdialog(40,85,280,300,[
button(250,40,20,60,'Save'),
button(250,200,20,60,'Quit'),
text(10,10,64,280,'Please input target objects which map to abstract objects identified in the dialogue:'),
text(90,45,16,100,'Source'),
text(115,40,16,100,O1),
text(136,40,16,100,O2),
text(157,40,16,100,O3),
text(178,40,16,100,O4),
text(199,40,16,100,O5),
text(90,170,16,100,'Target'),
edit(115,140,16,100,M1,T1),
edit(136,140,16,100,M2,T2),
edit(157,140,16,100,M3,T3),
edit(178,140,16,100,M4,T4),
edit(199,140,16,100,M5,T5)],Btn,
valid_objmappings(O1,O2,O3,O4,O5,T1,T2,T3,T4,T5)),
save_mappings(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp),
mapobjects_dialogue(Acp).

/* Rules to retrieve and set up ACP objects and mappings on the original
   screen. Each rule is determined by the number of objects available
   for a given ACP, so initially a findall counts the descriptors to select
   the appropriate builder rule. */

/* Rules to list ACP objects on the screen (range 2-5 objects per slot). */

getobjects(Olist,Acp) :-
findall(Object,acp_object(Object,Acp),Tlist),
length(Tlist,T),T=2,
Tlist=[O1,O2], Olist=[O1,O2,'','',''],!.

getobjects(Olist,Acp) :-
findall(Object,acp_object(Object,Acp),Tlist),
length(Tlist,T),T=3,
Tlist=[O1,O2,O3], Olist=[O1,O2,O3,'',''],!.

getobjects(Olist,Acp) :-
findall(Object,acp_object(Object,Acp),Tlist),
length(Tlist,T),T=4,
Tlist=[O1,O2,O3], Olist=[O1,O2,O3,O4,''],!.

getobjects(Olist,Acp) :-
findall(Object,acp_object(Object,Acp),Olist),
length(Tlist,T),T=5.

/* Rules to list mappings on the screen. */

getmappings(Mlist,Olist,Acp) :-
Olist=[O1,O2,O3,O4,O5],
getmapping1(O1,M1,Acp),
getmapping2(O2,M2,Acp),
getmapping3(O3,M3,Acp),
getmapping4(O4,M4,Acp),
```

```
getmapping5(O5,M5,Acp),
Mlist=[M1,M2,M3,M4,M5].

getmapping1(O1,M1,Acp) :-
rec_objectmatch(M1,O1,Score,Acp),
Score >= 1000,!.

getmapping1(O1,M1,Acp) :- M1='',!.

getmapping2(O2,M2,Acp) :-
rec_objectmatch(M2,O2,Score,Acp),
Score >= 1000,!.

getmapping2(O2,M2,Acp) :- M2='',!.

getmapping3(O3,M3,Acp) :-
rec_objectmatch(M3,O3,Score,Acp),
Score >= 1000,!.

getmapping3(O3,M3,Acp) :- M3='',!.

getmapping4(O4,M4,Acp) :-
rec_objectmatch(M4,O4,Score,Acp),
Score >= 1000,!.

getmapping4(O4,M4,Acp) :- M4='',!.

getmapping5(O5,M5,Acp) :-
rec_objectmatch(M5,O5,Score,Acp),
Score >= 1000,!.

getmapping5(O5,M5,Acp) :- M5='',!.

/* Validation rules to control data input. Checks ensure that objects not
   input for blank space objects, that input target objects exist, and target
   objects are not input for several different mappings. There is no need
   for any other more complex controls, since analysts can input any
   combination of mappings, and the subsequent screen and saving
   mechanisms cater for all possible events from analyst inputs. */

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
O1 = '',T1 =\= '',
beep(60), message(['Input objects must have corresponding objects: - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
O2 = '',T2 =\= '',
beep(60), message(['Input objects must have corresponding objects: - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
O3 = '',T3 =\= '',
beep(60), message(['Input objects must have corresponding objects: - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
O4 = '',T4 =\= '',
beep(60), message(['Input objects must have corresponding objects: - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
O5 = '',T5 =\= '',
beep(60), message(['Input objects must have corresponding objects: - Please try again']),!,fail.
```

```
valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
T1 =\= '',not target_object(T1),
beep(60), message(['The object',T1,'does not exist - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
T2 =\= '',not target_object(T2),
beep(60), message(['The object',T2,'does not exist - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
T3 =\= '',not target_object(T3),
beep(60), message(['The object',T3,'does not exist - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
T4 =\= '',not target_object(T4),
beep(60), message(['The object',T4,'does not exist - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :-
T5 =\= '',not target_object(T5),
beep(60), message(['The object',T5,'does not exist - Please try again']),!,fail.

valid_objmappings(D,B,O1,O2,O3,O4,O5,T1,T2,T3,T4,T5) :- !.

/* Rules to record valid object mappings: There are 7 branches to
   process for each of 4 lines of input:
   1- nothing before, nothing now, so ignore (no saves!!),
   2- score=1000 before, nothing now, delete predicate,
   3- score>1000 before, nothing now, subtract1000,delete & assert,
   4- something before, same now, so leave alone (no saves!!),
   5- nothing before, something now, & no existing mapping as result
      of matching, so create rule with score 1000,
   6- nothing before, something now, however previous predicate existed
      from matching, so add 1000 to score and change predicate,
   7- something before, changed now, a secondary level of rules is
      employed to process the two possible events for the old mapping,
      and the two possible events for the new mapping. */

save_mappings(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
findall(Acp,save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp),List).

/* Set of four rules to fulfil condition-2. */

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M1,O1,Score,_),Score=1000,T1='',
delete_mapping(M1,O1,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M2,O2,Score,_),Score=1000,T2='',
delete_mapping(M2,O2,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M3,O3,Score,_),Score=1000,T3='',
delete_mapping(M3,O3,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M4,O4,Score,_),Score=1000,T4='',
delete_mapping(M4,O4,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
```

```
rec_objectmatch(M5,O5,Score,_),Score=1000,T5='',
delete_mapping(M5,O5,Score,Acp).

/* Set of four rules to fulfil condition-3. */

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M1,O1,Score,_),Score>1000,T1='',M1=\='',
subtract_mapping(M1,O1,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M2,O2,Score,_),Score>1000,T2='',M2=\='',
subtract_mapping(M2,O2,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M3,O3,Score,_),Score>1000,T3='',M3=\='',
subtract_mapping(M3,O3,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M4,O4,Score,_),Score>1000,T4='',M4=\='',
subtract_mapping(M4,O4,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(M5,O5,Score,_),Score>1000,T5='',M5=\='',
subtract_mapping(M5,O5,Score,Acp).

/* Set of four rules to fulfil condition-5. */

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
not rec_objectmatch(T1,O1,Score,_),T1=\='',M1='',
create_mapping(T1,O1,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
not rec_objectmatch(T2,O2,Score,_),T2=\='',M2='',
create_mapping(T2,O2,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
not rec_objectmatch(T3,O3,Score,_),T3=\='',M3='',
create_mapping(T3,O3,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
not rec_objectmatch(T4,O4,Score,_),T4=\='',M4='',
create_mapping(T4,O4,Score,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
not rec_objectmatch(T5,O5,Score,_),T5=\='',M5='',
create_mapping(T5,O5,Score,Acp).

/* Set of four rules to fulfil condition-6. */

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(T1,O1,Oldscore,_),Oldscore<1000,M1='',T1=\='',
add_mapping(T1,O1,Oldscore,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(T2,O2,Oldscore,_),Oldscore<1000,M2='',T2=\='',
add_mapping(T2,O2,Oldscore,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(T3,O3,Oldscore,_),Oldscore<1000,M3='',T3=\='',
```

```
add_mapping(T3,O3,Oldscore,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(T4,O4,Oldscore,_),Oldscore<1000,M4='',T4=\='',
add_mapping(T4,O4,Oldscore,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
rec_objectmatch(T5,O5,Oldscore,_),Oldscore<1000,M5='',T5=\='',
add_mapping(T5,O5,Oldscore,Acp).

/* Set of four rules to fulfil condition-7. */

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T1=\='',M1=\='',M1=\=T1,
process_oldmapping(O1,M1,T1,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T1=\='',M1=\='',M1=\=T1,
process_newmapping(O1,M1,T1,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T2=\='',M2=\='',M2=\=T2,
process_oldmapping(O2,M2,T2,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T2=\='',M2=\='',M2=\=T2,
process_newmapping(O2,M2,T2,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T3=\='',M3=\='',M3=\=T3,
process_oldmapping(O3,M3,T3,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T3=\='',M3=\='',M3=\=T3,
process_newmapping(O3,M3,T3,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T4=\='',M4=\='',M4=\=T4,
process_oldmapping(O4,M4,T4,Acp).

save_mapping(O1,O2,O3,O4,O5,M1,M2,M3,M4,M5,T1,T2,T3,T4,T5,Acp) :-
T4=\='',M4=\='',M4=\=T4,
process_newmapping(O4,M4,T4,Acp).

/* Four updates to the rules necessary to process the conditions described
   in the above eight sections:
   1- delete a mapping,
   2- subtract 1000 from a score and modify mapping,
   3- create a mapping,
   4- add 1000 to score and modify mapping. */

delete_mapping(O1,O2,Score,Acp) :-
retract(rec_objectmatch(O1,O2,Score,_)),!.

subtract_mapping(O1,O2,Score,Acp) :-
rec_objectmatch(O1,O2,Oldscore,_),
Newscore is Oldscore-1000,
retract(rec_objectmatch(O1,O2,Oldscore,_)),
assertz(rec_objectmatch(O1,O2,Newscore,Acp)),!.
```

```
create_mapping(O1,O2,Score,Acp) :-
assertz(rec_objectmatch(O1,O2,1000,Acp)),!.

add_mapping(O1,O2,Oldscore,Acp) :-
rec_objectmatch(O1,O2,Oldscore,_),
Newscore is Oldscore+1000,
retract(rec_objectmatch(O1,O2,Oldscore,_)),
assertz(rec_objectmatch(O1,O2,Newscore,Acp)),!.

/* Two option-control rules to allow more complex processing during
   condition 7 . */

process_oldmapping(O1,M1,T1,Acp) :-
rec_objectmatch(M1,O1,Oldscore,_),
Oldscore=1000,delete_mapping(M1,O1,Oldscore,_),!.

process_oldmapping(O1,M1,T1,Acp) :-
rec_objectmatch(M1,O1,Oldscore,_),
Oldscore>1000,subtract_mapping(M1,O1,Oldscore,Acp),!.

process_newmapping(O1,M1,T1,Acp) :-
not rec_objectmatch(T1,O1,_,_),
create_mapping(T1,O1,_,Acp),!.

process_newmapping(O1,M1,T1,Acp) :-
rec_objectmatch(T1,O1,Score,_),
add_mapping(T1,O1,Score,Acp),!.
```

```
/* This window describes two programs which are called by all
   explanation windows to identify and present the most likely
   analogous mappings for each object in the source. The first program
   draws the mappings table on to the calling explanation window. The
   second program develops the mapping list from known analogous
   mappings. */

/* The first program is called by all windows. It draws the object
   list of mappings in the same position within each window. */

mappings_list(Win) :-
allmapping(Acp,Mappinglist),
add_pic(Win,maplist,[
brick(fillbox(25,287,118,196)),
blank(fillbox(28,320,16,130)),
text('Bookman',12,0,40,323,'Analogical Mappings'),
blank(fillbox(50,290,90,190)),
line((50,360),(140,360)),
line((65,290),(65,480)),
line((80,290),(80,480)),
line((95,290),(95,480)),
line((110,290),(110,480)),
line((125,290),(125,480))]),
displaymapping(Win,Mappinglist).

/* Series of subroutines to list all options for the mappings. */

displaymapping(Win,List) :- List=[],!.

displaymapping(Win,List) :-
length(List,T),T=1,
List=[(A1,A2,A3)],
add_pic(Win,maplist1,[
text('Bookman',12,0,61,295,A2),
text('Bookman',12,0,61,365,A1)]),!.

displaymapping(Win,List) :-
length(List,T),T=2,
List=[(A1,A2,A3),(B1,B2,B3)],
add_pic(Win,maplist2,[
text('Bookman',12,0,61,295,A2),
text('Bookman',12,0,61,365,A1),
text('Bookman',12,0,76,295,B2),
text('Bookman',12,0,76,365,B1)]),!.

displaymapping(Win,List) :-
length(List,T),T=3,
List=[(A1,A2,A3),(B1,B2,B3),(C1,C2,C3)],
add_pic(Win,maplist3,[
text('Bookman',12,0,61,295,A2),
text('Bookman',12,0,61,365,A1),
text('Bookman',12,0,76,295,B2),
text('Bookman',12,0,76,365,B1),
text('Bookman',12,0,91,295,C2),
text('Bookman',12,0,91,365,C1)]),!.

displaymapping(Win,List) :-
length(List,T),T=4,
List=[(A1,A2,A3),(B1,B2,B3),(C1,C2,C3),(D1,D2,D3)],
```

```
add_pic(Win,maplist4,[
text('Bookman',12,0,61,295,A2),
text('Bookman',12,0,61,365,A1),
text('Bookman',12,0,76,295,B2),
text('Bookman',12,0,76,365,B1),
text('Bookman',12,0,91,295,C2),
text('Bookman',12,0,91,365,C1),
text('Bookman',12,0,106,295,D2),
text('Bookman',12,0,106,365,D1)]),!.

displaymapping(Win,List) :-
length(List,T),T=5,
List=[(A1,A2,A3),(B1,B2,B3),(C1,C2,C3),(D1,D2,D3),(E1,E2,E3)],
add_pic(Win,maplist5,[
text('Bookman',12,0,61,295,A2),
text('Bookman',12,0,61,365,A1),
text('Bookman',12,0,76,295,B2),
text('Bookman',12,0,76,365,B1),
text('Bookman',12,0,91,295,C2),
text('Bookman',12,0,91,365,C1),
text('Bookman',12,0,106,295,D2),
text('Bookman',12,0,106,365,D1),
text('Bookman',12,0,121,295,E2),
text('Bookman',12,0,121,365,E1)]),!.

displaymapping(Win,List) :-
length(List,T),T=6,
List=[(A1,A2,A3),(B1,B2,B3),(C1,C2,C3),(D1,D2,D3),(E1,E2,E3),(F1,F2,F3)],
add_pic(Win,maplist6,[
text('Bookman',12,0,61,295,A2),
text('Bookman',12,0,61,365,A1),
text('Bookman',12,0,76,295,B2),
text('Bookman',12,0,76,365,B1),
text('Bookman',12,0,91,295,C2),
text('Bookman',12,0,91,365,C1),
text('Bookman',12,0,106,295,D2),
text('Bookman',12,0,106,365,D1),
text('Bookman',12,0,121,295,E2),
text('Bookman',12,0,121,365,E1),
text('Bookman',12,0,136,295,F2),
text('Bookman',12,0,136,365,F1)]),!.

/* The second program is called by the window drawing program to
   identify object mappings. Only the best candidate mapping for each
   abstract object is provided. The program is based around two
   possibilities:
   1- only mapping which exists for a source, so present that,
   2- several mappings exist for a source, so arrange in list, sort list, and
      take the top object.
   All object mappings come from valid rec_objectmatches. */

/* Rule to elicit the list of matches for each source object. */

allmapping(Acp,Mappinglist) :-
findall((Obj1,Obj2,Score),get_mappings(Obj1,Obj2,Score),Mappinglist).

/* Rules to identify specific mappings added to the list. The first version
   of the rule identifies single mappings by attempting to develop a list
   of other mappings with the same source object, and hopefully
```

failing. The second version of the rule identifies source objects with several candidate mappings. These mappings are added to a list and sorted by their score, so that the best mapping is at the head and list to be taken as the best match. In turn there are two versions of this second rule, to allow for ties on object scores, so that both objects are displayed on the explanation window. */

```
/* Simplest rule to identify a single mapping with a source object. */

get_mappings(Obj1,Obj2,Score) :-
rec_objectmatch(Obj1,Obj2,Score,_),
findall(Obj2,(
rec_objectmatch(Obj1,Obj2,_,_),
rec_objectmatch(Oth1,Obj2,_,_),
Obj1=\=Oth1),List),
List=[].

/* Rule to process the best match between several mappings with a
   source goal. */

get_mappings(Obj1,Obj2,Score) :-
rec_objectmatch(Obj1,Obj2,Score,_),
findall((S,O1,Obj2),many_mappings(O1,Obj2,S),Mlist),
length(Mlist,L),L>1,sort(Mlist,Nlist,[],1),
Nlist=[(Score,Obj1,Obj2),(Score2,_,_)|Rest],
Score>Score2.

/* Rule to process a tie between two good object scores. */

get_mappings(Obj1,Obj2,Score) :-
rec_objectmatch(Obj3,Obj2,Score,_),
rec_objectmatch(Obj4,Obj2,Score,_),
findall((S,O1,Obj2),many_mappings(O1,Obj2,S),Mlist),
length(Mlist,L),L>1,sort(Mlist,Nlist,[],1),
Nlist=[(Score,Obj3,Obj2),(Score2,Obj4,Obj2)|Rest],
Score=Score2,
concat(Obj3,' or ',Objtemp),
concat(Objtemp,Obj4,Obj1).

/* Subrule required for findall in both second versions of the rule. */

many_mappings(O1,Obj2,S) :-
rec_objectmatch(Obj1,Obj2,Score,_),
rec_objectmatch(O1,Obj2,S,_),
O1=\=Obj1.
```

# Descriptions of the Routines to Graphically Describe the Target Domain

```
/* This window models the target problem to help the analyst to better
   understand their domain. There are two windows - the first window
   called lists all target facts using the basic notation, whilst a second
   window attempts to model this description in an explanation-like
   format to better assist the analyst to understand his description of the
   target. Problem drawing program-1 describes the target-list program,
   while Problem drawing program-2 describes the target picture program.
   */

/* Call to the Window used in all relevant windows. */

see_target(double,Win) :-
disable_menu('Objects'),
disable_menu('Other Inputs'),
target_list('Target Domain').

/* Additional routine included to show target from the CONTROL menu,
   when the menu must be disabled. */

menu_target :-
disable_menu('Control'),
see_target(double,Win).

/* Definition of the List Window called by all target windows. */

target_list(Win) :-
wgcreate(Win,40,0,440,570,0,0,0,1,0),
setup_targetlist(Win),
gviewer(Win,off),
wfront(Win).

setup_targetlist(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
targetlist_quit(textbox('Chicago',12,0,8,0,32,32,1,'Return')),
targetlist_pic(textbox('Chicago',12,0,6,0,32,32,1,'Picture Target'))],1),
add_pic(Win,listheads,[
textline('Bookman',12,1,5,120,'Target Domain'),
textline('Bookman',10,1,20,5,'Structure of the target domain:'),
textline('Bookman',10,1,90,5,'Functions in the target domain:'),
textline('Bookman',10,1,160,5,'Object categories in the target domain:'),
textline('Bookman',10,1,194,5,'Conditions on Functions in the target domain:'),
textline('Bookman',10,1,240,5,'Requirements of the target system:'),
textline('Bookman',10,1,286,5,'Functions not initiated by the information system:'),
textline('Bookman',10,1,320,5,'Physical attributes of objects in the target domain:'),
textline('Bookman',10,1,390,5,'Labels describing the target system:')]),
list_structure(Win),
list_movement(Win),
list_property(Win),
list_condition(Win),
list_requirement(Win),
list_scope(Win),
list_physical(Win),
list_label(Win).

/* Definition of the Quit from the List Window */

targetlist_quit(double,Win) :-
```

```
check_control,
enable_menu('Objects'),
enable_menu('Other Inputs'),
wkill('Target Domain').

check_control :-
get_prop(menu,control,on),
set_prop(menu,control,off),
enable_menu('Control'),!.

check_control :- !.

targetlist_pic(double,Win) :-
target_problem('Target Problem').

/* Definitions of list rules which construct a listing of the target domain.
   Due to the possible length of specific descriptors in the problem we
   have used a findall and list length check to simplify the processing of
   this program, even if the resulting code looks a bit unwieldy. */

/* Write up to 8 options for the static target structure. */

list_structure(Win) :-
findall((O1,O2,R),target_sdata(O1,O2,R),Tlist),
list_structures(Win,Tlist).

list_structures(Win,Tlist) :-
length(Tlist,0),!.

list_structures(Win,Tlist) :-
length(Tlist,1),Tlist=[(A1,B1,R1)],
write_structure(S1,A1,B1,R1),
add_pic(Win,ls1,[
textline('Bookman',10,0,32,10,S1)]).

list_structures(Win,Tlist) :-
length(Tlist,2),Tlist=[(A1,B1,R1),(A2,B2,R2)],
write_structure(S1,A1,B1,R1),
write_structure(S2,A2,B2,R2),
add_pic(Win,ls2,[
textline('Bookman',10,0,32,10,S1),
textline('Bookman',10,0,44,10,S2)]).

list_structures(Win,Tlist) :-
length(Tlist,3),Tlist=[(A1,B1,R1),(A2,B2,R2),(A3,B3,R3)],
write_structure(S1,A1,B1,R1),
write_structure(S2,A2,B2,R2),
write_structure(S3,A3,B3,R3),
add_pic(Win,ls3,[
textline('Bookman',10,0,32,10,S1),
textline('Bookman',10,0,44,10,S2),
textline('Bookman',10,0,56,10,S3)]).

list_structures(Win,Tlist) :-
length(Tlist,4),Tlist=[(A1,B1,R1),(A2,B2,R2),(A3,B3,R3),(A4,B4,R4)],
write_structure(S1,A1,B1,R1),
write_structure(S2,A2,B2,R2),
write_structure(S3,A3,B3,R3),
write_structure(S4,A4,B4,R4),
```

```
add_pic(Win,ls4,[
textline('Bookman',10,0,32,10,S1),
textline('Bookman',10,0,44,10,S2),
textline('Bookman',10,0,56,10,S3),
textline('Bookman',10,0,68,10,S4)]).

list_structures(Win,Tlist) :-
length(Tlist,5),Tlist=[(A1,B1,R1),(A2,B2,R2),(A3,B3,R3),(A4,B4,R4),
(A5,B5,R5)],
write_structure(S1,A1,B1,R1),
write_structure(S2,A2,B2,R2),
write_structure(S3,A3,B3,R3),
write_structure(S4,A4,B4,R4),
write_structure(S5,A5,B5,R5),
add_pic(Win,ls5,[
textline('Bookman',10,0,32,10,S1),
textline('Bookman',10,0,44,10,S2),
textline('Bookman',10,0,56,10,S3),
textline('Bookman',10,0,68,10,S4),
textline('Bookman',10,0,32,250,S5)]).

list_structures(Win,Tlist) :-
length(Tlist,6),Tlist=[(A1,B1,R1),(A2,B2,R2),(A3,B3,R3),(A4,B4,R4),
(A5,B5,R5),(A6,B6,R6)],
write_structure(S1,A1,B1,R1),
write_structure(S2,A2,B2,R2),
write_structure(S3,A3,B3,R3),
write_structure(S4,A4,B4,R4),
write_structure(S5,A5,B5,R5),
write_structure(S6,A6,B6,R6),
add_pic(Win,ls6,[
textline('Bookman',10,0,32,10,S1),
textline('Bookman',10,0,44,10,S2),
textline('Bookman',10,0,56,10,S3),
textline('Bookman',10,0,68,10,S4),
textline('Bookman',10,0,32,250,S5),
textline('Bookman',10,0,44,250,S6)]).

list_structures(Win,Tlist) :-
length(Tlist,7),Tlist=[(A1,B1,R1),(A2,B2,R2),(A3,B3,R3),(A4,B4,R4),
(A5,B5,R5),(A6,B6,R6),(A7,B7,R7)],
write_structure(S1,A1,B1,R1),
write_structure(S2,A2,B2,R2),
write_structure(S3,A3,B3,R3),
write_structure(S4,A4,B4,R4),
write_structure(S5,A5,B5,R5),
write_structure(S6,A6,B6,R6),
write_structure(S7,A7,B7,R7),
add_pic(Win,ls7,[
textline('Bookman',10,0,32,10,S1),
textline('Bookman',10,0,44,10,S2),
textline('Bookman',10,0,56,10,S3),
textline('Bookman',10,0,68,10,S4),
textline('Bookman',10,0,32,250,S5),
textline('Bookman',10,0,44,250,S6),
textline('Bookman',10,0,56,250,S7)]).

list_structures(Win,Tlist) :-
length(Tlist,8),Tlist=[(A1,B1,R1),(A2,B2,R2),(A3,B3,R3),(A4,B4,R4),
```

```
(A5,B5,R5),(A6,B6,R6),(A7,B7,R7),(A8,B8,R8)],
write_structure(S1,A1,B1,R1),
write_structure(S2,A2,B2,R2),
write_structure(S3,A3,B3,R3),
write_structure(S4,A4,B4,R4),
write_structure(S5,A5,B5,R5),
write_structure(S6,A6,B6,R6),
write_structure(S7,A7,B7,R7),
write_structure(S8,A8,B8,R8),
add_pic(Win,ls8,[
textline('Bookman',10,0,32,10,S1),
textline('Bookman',10,0,44,10,S2),
textline('Bookman',10,0,56,10,S3),
textline('Bookman',10,0,68,10,S4),
textline('Bookman',10,0,32,250,S5),
textline('Bookman',10,0,44,250,S6),
textline('Bookman',10,0,56,250,S7),
textline('Bookman',10,0,68,250,S8)]).

/* Routines to output the movements in the target domain. */

list_movement(Win) :-
findall((T,O1,O2,O3,R),target_ddata(T,O1,O2,O3,R),Tlist),
list_movements(Win,Tlist).

list_movements(Win,Tlist) :-
length(Tlist,0),!.

list_movements(Win,Tlist) :-
length(Tlist,1),Tlist=[(T1,A1,B1,C1,R1)],
write_movement(S1,T1,A1,B1,C1,R1),
add_pic(Win,lm1,[
textline('Bookman',10,0,102,10,S1)]).

list_movements(Win,Tlist) :-
length(Tlist,2),Tlist=[(T1,A1,B1,C1,R1),(T2,A2,B2,C2,R2)],
write_movement(S1,T1,A1,B1,C1,R1),
write_movement(S2,T2,A2,B2,C2,R2),
add_pic(Win,lm2,[
textline('Bookman',10,0,102,10,S1),
textline('Bookman',10,0,114,10,S2)]).

list_movements(Win,Tlist) :-
length(Tlist,3),Tlist=[(T1,A1,B1,C1,R1),(T2,A2,B2,C2,R2),
(T3,A3,B3,C3,R3)],
write_movement(S1,T1,A1,B1,C1,R1),
write_movement(S2,T2,A2,B2,C2,R2),
write_movement(S3,T3,A3,B3,C3,R3),
add_pic(Win,lm3,[
textline('Bookman',10,0,102,10,S1),
textline('Bookman',10,0,114,10,S2),
textline('Bookman',10,0,126,10,S3)]).

list_movements(Win,Tlist) :-
length(Tlist,4),Tlist=[(T1,A1,B1,C1,R1),(T2,A2,B2,C2,R2),
(T3,A3,B3,C3,R3),(T4,A4,B4,C4,R4)],
write_movement(S1,T1,A1,B1,C1,R1),
write_movement(S2,T2,A2,B2,C2,R2),
write_movement(S3,T3,A3,B3,C3,R3),
```

```
write_movement(S4,T4,A4,B4,C4,R4),
add_pic(Win,lm4,[
textline('Bookman',10,0,102,10,S1),
textline('Bookman',10,0,114,10,S2),
textline('Bookman',10,0,126,10,S3),
textline('Bookman',10,0,138,10,S4)]).

/* Routines to output the properties of objects in the target domain. */

list_property(Win) :-
findall((O,P),target_pdata(O,P),Tlist),
list_properties(Win,Tlist).

list_properties(Win,Tlist) :-
length(Tlist,0),!.

list_properties(Win,Tlist) :-
length(Tlist,1),Tlist=[(O1,P1)],
write_properties(S1,O1,P1),
add_pic(Win,lp1,[
textline('Bookman',10,0,172,10,S1)]).

list_properties(Win,Tlist) :-
length(Tlist,2),Tlist=[(O1,P1),(O2,P2)],
write_properties(S1,O1,P1),
write_properties(S2,O2,P2),
add_pic(Win,lp2,[
textline('Bookman',10,0,172,10,S1),
textline('Bookman',10,0,172,250,S2)]).

/* Routines to output the conditions in the target domain. */

list_condition(Win) :-
findall((F,Cond),
target_cdata(F,Cond),Tlist),
list_conditions(Win,Tlist).

list_conditions(Win,Tlist) :-
length(Tlist,0),!.

list_conditions(Win,Tlist) :-
length(Tlist,1),Tlist=[(F,C)],
write_conditions(S1,F,C),
add_pic(Win,lc1,[
textline('Bookman',10,0,206,10,S1)]).

list_conditions(Win,Tlist) :-
length(Tlist,2),Tlist=[(F1,C1),(F2,C2)],
write_conditions(S1,F1,C1),
write_conditions(S2,F2,C2),
add_pic(Win,lc2,[
textline('Bookman',10,0,206,10,S1),
textline('Bookman',10,0,218,10,S2)]).

/* Write up to two system requirements. */

list_requirement(Win) :-
findall((O1,O2,R,''),target_reqt(O1,O2,R),L1),
findall((O3,O4,S,T),target_reqt(O3,O4,S,T),L2),
```

```
append(L1,L2,Tlist),
list_requirements(Win,Tlist).

list_requirements(Win,Tlist) :-
length(Tlist,0),!.

list_requirements(Win,Tlist) :-
length(Tlist,1),Tlist=[(A1,B1,R1,P1)],
write_requirements(S1,A1,B1,R1,P1),
add_pic(Win,lre1,[
textline('Bookman',10,0,252,10,S1)]).

list_requirements(Win,Tlist) :-
length(Tlist,2),Tlist=[(A1,B1,R1,P1),(A2,B2,R2,P2)],
write_requirements(S1,A1,B1,R1,P1),
write_requirements(S2,A2,B2,R2,P2),
add_pic(Win,lre2,[
textline('Bookman',10,0,252,10,S1),
textline('Bookman',10,0,264,10,S2)]).

/* Routines to output the movements beyond the scope of the target
   domain. */

list_scope(Win) :-
findall(Mvmt,target_scope(Mvmt),Tlist),
list_scopes(Win,Tlist).

list_scopes(Win,Tlist) :-
length(Tlist,0),!.

list_scopes(Win,Tlist) :-
length(Tlist,1),Tlist=[Mvmt1],
add_pic(Win,ls1,[
textline('Bookman',10,0,298,10,Mvmt1)]).

list_scopes(Win,Tlist) :-
length(Tlist,2),Tlist=[Mvmt1,Mvmt2],
add_pic(Win,ls2,[
textline('Bookman',10,0,298,10,Mvmt1),
textline('Bookman',10,0,310,10,Mvmt2)]).

/* Routines to output up to a maximum of 5 physical object properties. */

list_physical(Win) :-
findall((O,P),target_phyprop(O,P),Tlist),
list_physicals(Win,Tlist).

list_physicals(Win,Tlist) :-
length(Tlist,0),!.

list_physicals(Win,Tlist) :-
length(Tlist,1),Tlist=[(A1,P1)],
write_properties(S1,A1,P1),
add_pic(Win,lf1,[
textline('Bookman',10,0,344,10,S1)]).

list_physicals(Win,Tlist) :-
length(Tlist,2),Tlist=[(A1,P1),(A2,P2)],
write_properties(S1,A1,P1),
```

```
write_properties(S2,A2,P2),
add_pic(Win,lf2,[
textline('Bookman',10,0,344,10,S1),
textline('Bookman',10,0,356,10,S2)]).

list_physicals(Win,Tlist) :-
length(Tlist,3),Tlist=[(A1,P1),(A2,P2),(A3,P3)],
write_properties(S1,A1,P1),
write_properties(S2,A2,P2),
write_properties(S3,A3,P3),
add_pic(Win,lf3,[
textline('Bookman',10,0,344,10,S1),
textline('Bookman',10,0,356,290,S2),
textline('Bookman',10,0,368,10,S3)]).

list_physicals(Win,Tlist) :-
length(Tlist,4),Tlist=[(A1,P1),(A2,P2),(A3,P3),(A4,P4)],
write_properties(S1,A1,P1),
write_properties(S2,A2,P2),
write_properties(S3,A3,P3),
write_properties(S4,A4,P4),
add_pic(Win,lf4,[
textline('Bookman',10,0,344,10,S1),
textline('Bookman',10,0,356,10,S2),
textline('Bookman',10,0,368,10,S3),
textline('Bookman',10,0,344,250,S4)]).

list_physicals(Win,Tlist) :-
length(Tlist,5),Tlist=[(A1,P1),(A2,P2),(A3,P3),(A4,P4),(A5,P5)],
write_properties(S1,A1,P1),
write_properties(S2,A2,P2),
write_properties(S3,A3,P3),
write_properties(S4,A4,P4),
write_properties(S5,A5,P5),
add_pic(Win,lf5,[
textline('Bookman',10,0,344,10,S1),
textline('Bookman',10,0,356,10,S2),
textline('Bookman',10,0,368,10,S3),
textline('Bookman',10,0,344,250,S4),
textline('Bookman',10,0,356,250,S5)]).

/* Routines to output functions and labels. */

list_label(Win) :-
findall(L,target_label(L),List),
list_funcs(Win,List).

list_funcs(Win,List) :-
length(List,0),!.

list_funcs(Win,Tlist) :-
length(Tlist,1),Tlist=[A],
add_pic(Win,ll1,[
textline('Bookman',10,0,402,10,A)]).

list_funcs(Win,Tlist) :-
length(Tlist,2),Tlist=[A,B],
add_pic(Win,ll2,[
textline('Bookman',10,0,402,10,A),
```

```
textline('Bookman',10,0,414,10,B)]).

list_funcs(Win,Tlist) :-
length(Tlist,3),Tlist=[A,B,C],
add_pic(Win,ll3,[
textline('Bookman',10,0,402,10,A),
textline('Bookman',10,0,414,10,B),
textline('Bookman',10,0,402,230,C)]).

/* Routines to write the sentences for each target description, ie to
   organise the layout of each descriptive sentence depending upon the
   data described in the sentence. */

write_structure(Sentence1,Obj1,Obj2,Rel) :-
concat(Obj1,' ',A),
concat(A,Rel,B),
concat(B,' ',C),
concat(C,Obj2,Sentence1).

write_movement(Sentence2,Tran,Obj1,Obj2,Obj3,Rel) :-
concat(Tran,' - ',X),
concat(X,Rel,Y),
concat(Y,' ',A),
concat(A,Obj1,B),
concat(B,' from ',C),
concat(C,Obj2,D),
concat(D,' to ',E),
concat(E,Obj3,Sentence2).

write_properties(Sentence3,Obj,Property) :-
concat(Obj,' is ',A),
concat(A,Property,Sentence3).

write_conditions(Sentence4,Func,Cond) :-
concat(Func,' when ',A),
concat(A,Cond,Sentence4).

write_requirements(Sentence5,Obj1,Obj2,Rel,Prop) :-
write_structure(S1,Obj1,Obj2,Rel),
concat(S1,' with ',A),
concat(A,Prop,Sentence5).
```

```
/* Definition of the Second 'Picture' Window, which develops a model of
   the target problem to support the list of the target problem. */

target_problem('Target Problem') :-
wgcreate('Target Problem',40,250,250,320,0,0,0,1,0),
setup_targetwin('Target Problem'),
findall('Target Problem',draw_problem('Target Problem'),Dlist),
gviewer('Target Problem',off),
wfront('Target Problem').

setup_targetwin(Win) :-
gsplit(Win,70),
gcursor(Win,hand),
add_tools(Win,[
targetpic_quit(textbox('Chicago',12,0,8,0,32,32,1,'Return'))],1).

/* Definition of the Quit from the Picture Window. */

targetpic_quit(double,Win) :-
wkill('Target Problem').

/* The 'draw-problem' rule identifies input parts of the problem and draws
   them on the above window. A considerable number of such rules are
   required to properly model likely descriptions of the target problem.
   The drawing is separated from the rules which decide the drawing
   of objects, so to simplify the problem and permit simpler modification
   of the program as necessary. Do not include object properties at the
   present */

draw_problem(Win) :-
target_name(Name),
add_pic(Win,w1,[
textline('Bookman',12,2,5,20,Name),
textbox('Bookman',11,0,195,5,36,200,0,'Ira is sorry that it may be unable to construct a complete model of
the target domain')]).

/* Add objects to the world */

draw_problem(Win) :-
singobj_inspace(Object),
add_pic(Win,w1a,[
fillbox(39,40,13,13),
textline('Bookman',10,2,29,20,Object)]).

draw_problem(Win) :-
singobjs_inspace(Object1,Object2),
add_pic(Win,w1b,[
fillbox(179,180,13,13),
textline('Bookman',10,2,169,160,Object1),
fillbox(39,40,13,13),
textline('Bookman',10,2,29,20,Object2)]).

draw_problem(Win) :-
mulobj_inspace(Object),
add_pic(Win,w1c,[
fillbox(31,51,13,13),
fillbox(45,65,13,13),
text('Bookman',10,2,25,34,Object)]).
```

```
draw_problem(Win) :-
mulobjs_inspace(Object1,Object2),
add_pic(Win,w1d,[
fillbox(31,51,13,13),
fillbox(45,65,13,13),
text('Bookman',10,2,25,34,Object1),
fillbox(181,201,13,13),
fillbox(195,215,13,13),
text('Bookman',10,2,175,184,Object2)]).

/* Add the slots to the problem space */

draw_problem(Win) :-
oneslot_inspace(Object),
add_pic(Win,w2a,[
speckled(fillbox(80,80,90,90)),
text('Bookman',10,2,74,150,Object)]).

draw_problem(Win) :-
mulslot_inspace(Object),
add_pic(Win,w2b,[
speckled(fillbox(80,80,45,45)),
speckled(fillbox(130,130,45,45)),
text('Bookman',10,2,125,165,Object),
text('Bookman',10,2,75,90,Object)]).

/* Add the second layer of slots to the diagram - four rules to add one or
   many slots to the single slot or many slots for the system */

draw_problem(Win) :-
oneslot_inoneslot(Object),
add_pic(Win,w3a,[
blank(fillbox(110,110,30,30)),
text('Bookman',10,2,105,120,Object)]).

draw_problem(Win) :-
manyslot_inoneslot(Object),
add_pic(Win,w3b,[
blank(fillbox(82,82,26,26)),
blank(fillbox(82,112,26,26)),
blank(fillbox(82,142,26,26)),
blank(fillbox(112,82,26,26)),
blank(fillbox(112,112,26,26)),
blank(fillbox(112,142,26,26)),
blank(fillbox(142,82,26,26)),
blank(fillbox(142,112,26,26)),
blank(fillbox(142,142,26,26)),
line((154,168),(154,190)),
text('Bookman',10,2,153,191,Object)]).

draw_problem(Win) :-
oneslot_inmanyslot(Object),
add_pic(Win,w3c,[
blank(fillbox(87,87,30,30)),
blank(fillbox(137,137,30,30)),
line((117,95),(139,95)),
line((137,145),(125,145)),
text('Bookman',10,2,150,90,Object),
text('Bookman',10,2,125,140,Object)]).
```

```
draw_problem(Win) :-
manyslot_inmanyslot(Object),
add_pic(Win,w4c,[
blank(fillbox(82,82,20,20)),
blank(fillbox(103,103,20,20)),
blank(fillbox(132,132,20,20)),
blank(fillbox(153,153,20,20)),
text('Bookman',10,2,110,125,Object),
text('Bookman',10,2,160,175,Object)]).

/* Add Objects to the 12 possible combinations of objects in the
   structure (one object or many objects) -
   1 - single slot in space,
   2 - several slots in space,
   3 - single slot in single slot,
   4 - several slots in single slot,
   5 - single slot in several slots,
   6 - several slots in several slots. The program is described in
   corresponding pairs. */

/* 1----------*/

draw_problem(Win) :-
oneobj_inoneslot(Object),
add_pic(Win,w5c,[
fillbox(120,120,13,13),
text('Bookman',10,2,115,110,Object)]).

draw_problem(Win) :-
manyobj_inoneslot(Object),
add_pic(Win,w6c,[
fillbox(100,105,13,13),
fillbox(120,120,13,13),
text('Bookman',10,2,115,120,Object)]).

/* 2----------*/

draw_problem(Win) :-
oneobj_inmanyslot(Object),
add_pic(Win,w7c,[
fillbox(96,96,13,13),
fillbox(146,146,13,13),
text('Bookman',10,2,100,110,Object),
text('Bookman',10,2,150,160,Object)]).

draw_problem(Win) :-
manyobj_inmanyslot(Object),
add_pic(Win,w8c,[
fillbox(85,85,13,13),
fillbox(107,107,13,13),
fillbox(135,135,13,13),
fillbox(157,157,13,13),
text('Bookman',10,2,91,98,Object),
text('Bookman',10,2,165,170,Object)]).

/* 3----------*/

draw_problem(Win) :-
```

```
oneobj_inoneslotslot(Object),
add_pic(Win,w9c,[
fillbox(118,118,13,13),
text('Bookman',10,2,125,131,Object)]).

draw_problem(Win) :-
manyobj_inoneslotslot(Object),
add_pic(Win,w10c,[
fillbox(112,112,13,13),
fillbox(125,125,13,13),
text('Bookman',10,2,120,125,Object)]).

/* 4----------*/

draw_problem(Win) :-
oneobj_inmanyslotoneslot(Object),
add_pic(Win,w11c,[
fillbox(88,88,13,13),
fillbox(118,118,13,13),
fillbox(88,148,13,13),
line((94,161),(94,180)),
text('Bookman',10,2,88,178,Object)]).

draw_problem(Win) :-
manyobj_inmanyslotoneslot(Object),
add_pic(Win,w12c,[
fillbox(84,84,13,13),
fillbox(93,93,13,13),
fillbox(114,114,13,13),
fillbox(123,123,13,13),
fillbox(144,84,13,13),
fillbox(153,93,13,13),
text('Bookman',10,2,90,97,Object)]).

/* 5--------*/

draw_problem(Win) :-
oneobj_inoneslotmanyslot(Object),
add_pic(Win,w13c,[
fillbox(95,95,13,13),
fillbox(145,145,13,13),
text('Bookman',10,2,150,158,Object)]).

draw_problem(Win) :-
manyobj_inoneslotmanyslot(Object),
add_pic(Win,w14c,[
fillbox(89,89,13,13),
fillbox(102,102,13,13),
fillbox(139,139,13,13),
fillbox(152,152,13,13),
text('Bookman',10,2,94,102,Object),
text('Bookman',10,2,157,165,Object)]).

/* 6---------- to do */
```

```
/* The first series of rules identifies the requirements to add objects to
   the problem space, and slots to the object space. They are called from
   programs which draw the concepts onto the picture window */

/* Adding a single object to the picture. Two sets of rules are needed to
   cater for both sets of possible objects. */

singobj_inspace(Object) :-
findall(Objects,singobject_inspace(Objects),Olist),
length(Olist,1),Olist=[Object].

singobjs_inspace(Object1,Object2) :-
findall(Objects,singobject_inspace(Objects),Olist),
length(Olist,2),Olist=[Object1,Object2].

singobject_inspace(Object) :-
target_sdata(world,Object,has_one),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* Add many objects to the picture. Two rules are needed, to account for
   one set of objects or two sets of objects. */

mulobj_inspace(Object) :-
findall(Objects,mulobject_inspace(Objects),Olist),
length(Olist,1),Olist=[Object].

mulobjs_inspace(Object1,Object2) :-
findall(Objects,mulobject_inspace(Objects),Olist),
length(Olist,2),Olist=[Object1,Object2].

mulobject_inspace(Object) :-
target_sdata(world,Object,has_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* Adding a single slot to the space */

oneslot_inspace(Slot) :-
target_sdata(world,Slot,has_one),
target_sdata(Slot,_,contains_one).

oneslot_inspace(Slot) :-
target_sdata(world,Slot,has_one),
target_sdata(Slot,_,contains_many).

/* Adding many slots to the space */

mulslot_inspace(Slot) :-
target_sdata(world,Slot,has_many),
target_sdata(Slot,_,contains_one).

mulslot_inspace(Slot) :-
target_sdata(world,Slot,has_many),
target_sdata(Slot,_,contains_many).

/* Rules to add second layer of slots to the diagram, which tend to
   overwrite the original slots where necessary. Two rules for each case,
   to ensure that the Slot2s contain something - so they are slots and not
```

```
   objects */

  oneslot_inoneslot(Slot2) :-
  target_sdata(world,Slot1,has_one),
  target_sdata(Slot1,Slot2,contains_one),
  target_sdata(Slot2,_,contains_one).

  oneslot_inoneslot(Slot2) :-
  target_sdata(world,Slot1,has_one),
  target_sdata(Slot1,Slot2,contains_one),
  target_sdata(Slot2,_,contains_many).

  manyslot_inoneslot(Slot2) :-
  target_sdata(world,Slot1,has_one),
  target_sdata(Slot1,Slot2,contains_many),
  target_sdata(Slot2,_,contains_one).

  manyslot_inoneslot(Slot2) :-
  target_sdata(world,Slot1,has_one),
  target_sdata(Slot1,Slot2,contains_many),
  target_sdata(Slot2,_,contains_many).

  oneslot_inmanyslot(Slot2) :-
  target_sdata(world,Slot1,has_many),
  target_sdata(Slot1,Slot2,contains_one),
  target_sdata(Slot2,_,contains_one).

  oneslot_inmanyslot(Slot2) :-
  target_sdata(world,Slot1,has_many),
  target_sdata(Slot1,Slot2,contains_one),
  target_sdata(Slot2,_,contains_many).

  manyslot_inmanyslot(Slot2) :-
  target_sdata(world,Slot1,has_many),
  target_sdata(Slot1,Slot2,contains_many),
  target_sdata(Slot2,_,contains_one).

  manyslot_inmanyslot(Slot2) :-
  target_sdata(world,Slot1,has_many),
  target_sdata(Slot1,Slot2,contains_many),
  target_sdata(Slot2,_,contains_many).

/* Rules to add objects to the drawn slots. They are numbered one
   to six, in order to link them to the document support about the nature
   of such diagrams. For each rule there are two instances, to identify
   contains_one and contains_many. */

/* 1a----------*/

oneobj_inoneslot(Object) :-
target_sdata(world,Slot,has_one),
target_sdata(Slot,Object,contains_one),
not target_sdata(Slot,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 1b----------*/

manyobj_inoneslot(Object) :-
```

```
target_sdata(world,Slot,has_one),
target_sdata(Slot,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 2a----------*/

oneobj_inmanyslot(Object) :-
target_sdata(world,Slot,has_many),
target_sdata(Slot,Object,contains_one),
not target_sdata(Slot,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 2b----------*/

manyobj_inmanyslot(Object) :-
target_sdata(world,Slot,has_many),
target_sdata(Slot,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 3a----------*/

oneobj_inoneslotslot(Object) :-
target_sdata(world,Slot1,has_one),
target_sdata(Slot1,Slot2,contains_one),
target_sdata(Slot2,Object,contains_one),
not target_sdata(Slot,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 3b----------*/

manyobj_inoneslotslot(Object) :-
target_sdata(world,Slot1,has_one),
target_sdata(Slot1,Slot2,contains_one),
target_sdata(Slot2,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 4a----------*/

oneobj_inmanyslotoneslot(Object) :-
target_sdata(world,Slot1,has_one),
target_sdata(Slot1,Slot2,contains_many),
target_sdata(Slot2,Object,contains_one),
not target_sdata(Slot2,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 4b----------*/

manyobj_inmanyslotoneslot(Object) :-
target_sdata(world,Slot1,has_one),
target_sdata(Slot1,Slot2,contains_many),
target_sdata(Slot2,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).
```

```
/* 5a----------*/

oneobj_inoneslotmanyslot(Object) :-
target_sdata(world,Slot1,has_many),
target_sdata(Slot1,Slot2,contains_one),
target_sdata(Slot2,Object,contains_one),
not target_sdata(Slot2,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 5b----------*/

manyobj_inoneslotmanyslot(Object) :-
target_sdata(world,Slot1,has_many),
target_sdata(Slot1,Slot2,contains_one),
target_sdata(Slot2,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 6a----------*/

oneobj_inmanyslotmanyslot(Object) :-
target_sdata(world,Slot1,has_many),
target_sdata(Slot1,Slot2,contains_many),
target_sdata(Slot2,Object,contains_one),
not target_sdata(Slot2,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).

/* 6b----------*/

manyobj_inmanyslotmanyslot(Object) :-
target_sdata(world,Slot1,has_many),
target_sdata(Slot1,Slot2,contains_many),
target_sdata(Slot2,Object,contains_many),
not target_sdata(Object,_,contains_one),
not target_sdata(Object,_,contains_many).
```

# Description of Other Useful Routines

```
/* The general help routine accessible by every window. The first access
   creates the window, then subsequent accesses call the window up from
   closed position using 'wfront'. There are a couple of buffer rules
   (openhelp) and (showhelp) to ensure that the rules fire without
   outputting the standard error messages associated with window rules. */

general_help(double,Win) :-
get_prop(help,window,0),
set_prop(help,window,1),
openhelp.

general_help(double,Win) :-
get_prop(help,window,1),
showhelp.

openhelp :-
help_window('General Help Window'),!.

showhelp :-
wfront('General Help Window'),!.

help_window(Win) :-
wgcreate(Win,130,72,300,300,0,0,300,1,1),
setup_genhelp(Win),
wfront(Win),
gviewer(Win,off),
gscroll_to(Win,-300,0).

setup_genhelp(Win) :-
gsplit(Win,0),
gcursor(Win,hand),
add_pic(Win,genhelp,[
textbox('Times',14,4,-295,5,32,250,1,'General Help'),
textbox('Times',12,0,-270,5,48,280,0,'General help gives an overview of the 10 windows for inputting
data about your new problem. Most windows identify facts about the problem domain while several later
windows request data about the computer system.'),
textbox('Times',12,0,-216,5,48,280,0,'During the early stages of data input you are encouraged to sketch
specific features of your problem on the paper provided. Descriptions of these sketches are used as a basis
for describing the problem domain.'),
textline('Times',12,2,-162,5,'1- Introduction to Ira'),
textbox('Times',12,0,-150,5,24,280,0,'This window describes how to input data using Ira, and requests the
name and major goal of your new system.'),
textline('Times',12,2,-120,5,'2- Functions Window'),
textbox('Times',12,0,-108,5,24,280,0,'This window requests you to select up to four functions which
accurately describe the required system.'),
textline('Times',12,2,-78,5,'3- Function Defn and Structural Windows'),
textbox('Times',12,0,-66,5,24,280,0,'These two windows are displayed for each function entered in the
Functions Window:'),
textline('Times',12,2,-36,5,'3(a)- Function Definition Window'),
textbox('Times',12,0,-24,5,24,280,0,'Sketch the function using the terminology provided by the window,
then enter this definition using the dialogue provided.'),
textline('Times',12,2,6,5,'3(b)- Structural Window'),
textbox('Times',12,0,18,5,24,280,0,'Enter additional features about the structural relationships between
objects identified for that function.'),
textline('Times',12,2,48,5,'4- Structures Window'),
textbox('Times',12,0,60,5,36,280,0,'Expand and combine all your sketches of system functions, then enter
descriptions representing additional facts resulting from these changes.'),
textline('Times',12,2,102,5,'5- Categories Window'),
textbox('Times',12,0,114,5,12,280,0,'Categorise objects identified during previous windows.'),
```

```
textline('Times',12,2,132,5,'6- Conditions Window'),
textbox('Times',12,0,144,5,12,280,0,'Identify conditions under which system functions occur.'),
textline('Times',12,2,162,5,'7- Requirements Window'),
textbox('Times',12,0,174,5,48,280,0,'Select specific requirements to be achieved by the new system. These
requirements are described in terms of states to be achieved, represented as object-relations entered in earlier
windows.'),
textline('Times',12,2,228,5,'8- Scope Window'),
textbox('Times',12,0,240,5,24,280,0,'Identify each system as either initiated by the computer system or
responsive to events beyond the system.'),
textline('Times',12,2,270,5,'9- Labels Window'),
textbox('Times',12,0,282,5,12,280,0,'Select general terms which best describe the system.'),
textline('Times',12,2,300,5,'10- Physical Window'),
textbox('Times',12,0,312,5,24,280,0,'Select physical attributes which best describe the objects in your
problem application.'),
textline('Times',12,2,342,5,'10- Searching/Update Window'),
textbox('Times',12,0,354,5,48,280,0,'This final window gives you the option of using pulldown menus to
modify any descriptions of your problem or of matching your problem description to sofware engineering
problem types known to Ira.')]).


/* Routines to display complex structures on scroll menus, then
   match the selected menu item to that structure once control is
   returned to the machine.

   The get program concats each variable in a structure to develop a
   composite variable, T7.

   The find program concatenates each target rule, then matches the
   concatenated structure until it equal to the structure selected from
   the menu - it is quite simple really. */

get_ddata(T7) :-
target_ddata(O1,O2,O3,R),
concat(',',R,T1),
concat(O3,T1,T2),
concat(',',T2,T4),
concat(O2,T4,T5),
concat(',',T5,T6),
concat(O1,T6,T7).

get_sdata(T7) :-
target_sdata(O1,O2,R),
concat(',',R,T1),
concat(O2,T1,T5),
concat(',',T5,T6),
concat(O1,T6,T7).

find_ddata(O1,O2,O3,R,Selected) :-
target_ddata(O1,O2,O3,R),
concat(',',R,T1),
concat(O3,T1,T2),
concat(',',T2,T4),
concat(O2,T4,T5),
concat(',',T5,T6),
concat(O1,T6,T7),
compare(=,T7,Selected).

find_sdata(O1,O2,R,Selected) :-
target_sdata(O1,O2,R),
```

```
concat(',',R,T1),
concat(O2,T1,T5),
concat(',',T5,T6),
concat(O1,T6,T7),
compare(=,T7,Selected).

/* Subroutine to build the list of all objects, accessed by many programs
   to get the objects on to the screen without causing the dialogue to
   fail. This is achieved by constructing a newlist from the findall list,
   and filling in the spaces behind the objects so that the list always has
   four objects, albeit blank ones */

build_objects(Newlist) :-
findall(Objects,target_object(Objects),Oldlist),
get_objects(Oldlist,Newlist).

get_objects(Oldlist,Newlist) :-
length(Oldlist,1),
Oldlist = [O1],
Newlist = [O1,'','',''].
get_objects(Oldlist,Newlist) :-
length(Oldlist,2),
Oldlist = [O1,O2],
Newlist = [O1,O2,'',''].
get_objects(Oldlist,Newlist) :-
length(Oldlist,3),
Oldlist = [O1,O2,O3],
Newlist = [O1,O2,O3,''].
get_objects(Oldlist,Newlist) :-
length(Oldlist,4),
Oldlist = [O1,O2,O3,O4],
Newlist = [O1,O2,O3,O4,''].
get_objects(Oldlist,Newlist) :-
length(Oldlist,5),
Newlist = Oldlist.

/* The following routine is used to control input of objects, functions &
   labels, to ensure they begin with a small letter, and do not include any
   control characters. This is most easily achieved by stating what is
   allowed. There are 3 levels of control in the program. The first checks
   the first character, the second checks other characters. The third level
   is built into the stringof command and weeds out other unnecessary
   characters at an early stage. */

valid_character(Name) :-
stringof(Namelist,Name),
check_firstcharacter(Namelist),
check_characters(Namelist).

check_firstcharacter(Namelist) :-
Namelist=[Char|Restlist],
Smalletters=[a,b,c,d,e,f,g,h,i,j,
k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z],
on(Char,Smalletters),!.

check_characters([Char|[]]) :- !.
check_characters([Char|Namelist]) :-
Okletters=[a,b,c,d,e,f,g,h,i,j,
k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,
```

```
'A','B','C','D','E','F','G','H','I','J','K','L','M',
'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
'1','2','3','4','5','6','7','8','9','0','_'],
on(Char,Okletters),
check_characters(Namelist).

/* Generic routine used on a number of occasions to identify when an
   object is part of an existing structure, especially during maintenance of
   target knowledge consistency. It checks for existence in structure,
   movement, physical and properties. */

used_object(Object) :-
target_sdata(Object,_,_),!.

used_object(Object) :-
target_sdata(_,Object,_),!.

used_object(Object) :-
target_ddata(_,Object,_,_,_),!.

used_object(Object) :-
target_ddata(_,_,Object,_,_),!.

used_object(Object) :-
target_ddata(_,_,_,Object,_),!.

used_object(Object) :-
target_pdata(Object,_),!.

used_object(Object) :-
target_phyprop(Object,_),!.

/* The set counters routine ran at the beginning of a session with Ira. */

set_counters :-
set_prop(help,window,0),
set_prop(delete,condition,1).

/* Routine used by the second-pass ACP matches to construct a list of
   partially-fitting ACPs for the dialogue selections. */

get_fournames(Acplist,Newlist) :-
length(Acplist,1),
Acplist=[A],Newlist=[A,'','',''],!.

get_fournames(Acplist,Newlist) :-
length(Acplist,2),
Acplist=[A,B],Newlist=[A,B,'',''],!.

get_fournames(Acplist,Newlist) :-
length(Acplist,3),
Acplist=[A,B,C],Newlist=[A,B,C,''],!.

get_fournames(Acplist,Newlist) :-
length(Acplist,4),
Newlist=Acplist.
```