# CONCEPTUAL PROCESS MODELLING

**Note:**

- **This document explains how Conceptual Process Modelling is carried out within an SSADM framework.**

- **We hope that you will find this document useful. If you have any feedback/comments/queries etc, please email them to:**
  **walter.acuti@sandhill.co.uk**

- **For further information about SSADM, please visit our website at:**
  **www.sandhill.co.uk/pware/ssadm.htm**

**Table of Contents**

## INTRODUCTION

Conceptual Process Modelling comprises a number of techniques which are used to define the database processing which is triggered in response to events and enquiries.

The products which result from the Conceptual Process Modelling techniques are:

- **Enquiry Access Paths**

  These are used to validate the Required System Logical Data Model and to specify the database processing logic for each enquiry to be handled by the system.

- **Effect Correspondence Diagrams**

  These are used to validate the Required System Logical Data Model and to specify the database processing logic for each event which has to be handled by the system.

- **Enquiry Process Models**

  These are Jackson-like structures which are derived from Enquiry Access Paths and form the logical design specification for retrieval database processing for certain types of implementation environment.

- **Update Process Models**

  These are Jackson-like structures which are derived from Effect Correspondence Diagrams and form the logical design specification for update database processing for certain types of implementation environment.

## Enquiry Access Paths

Enquiry Access Paths define how the Required System Logical Data Model is navigated in order to retrieve the data necessary to generate the output for an enquiry. Enquiry Access Paths (EAPs) are used to achieve the following objectives:

- to validate the Required System Logical Data Model and in particular to show that the LDM can support the requirements of the enquiry processes;
- to provide a specification of the data accesses required for each enquiry processes
- to form the basis for logical and physical process models for certain implementation environments.

By producing EAPs, it is possible that changes will be needed to the emerging data and processing specifications for the new system. On the data side, producing EAPs may identify the need for new entities, attributes or relationships in order to support the enquiry processes - the Required System LDM must be modified to incorporate any such changes.

On the processing side, EAPs may identify the need for additional processes (eg to sort data) in order to output data in the format which Users require. The prime purpose of EAPs is to ensure that the data needed for output purposes can be retrieved from the Required System LDM. If the data is retrieved in a format which does not accord with how the data must be output, additional processing can be included to format the data as required, eg to sort, suppress duplicates, count and calculate derived values. Such processing can either be included as part of an EAP or as part of the relevant function.

During Logical Design (Stage 5), Enquiry Access Paths are transformed into Enquiry Process Models. These are Jackson-like structures which can provide a basis for physical process design for certain implementation environments, particularly those which use Jackson Programming techniques.

## Effect Correspondence Diagrams

Effect Correspondence Diagrams define how the Required System Logical Data Model is navigated and updated in order to retrieve data from and update the database in response to each event.

Effect Correspondence Diagrams (ECDs) are used to achieve the following objectives:

- to validate the Required System Logical Data Model and in particular to show that the LDM can support the requirements of each event process;
- to validate the results of Entity Life History Analysis;
- to provide a specification of the data accesses and data updates required for each event process;
- to form the basis for logical and physical process models for certain implementation environments.

An ECD is produced for each event and is primarily derived from the Entity Life Histories. Whereas an Entity Life History is drawn from an entity’s perspective and shows what happens to an entity over time in terms of all the events which can affect the entity, an ECD is drawn from an event’s perspective and shows:

- all the effects which an event has on entities;
- how these effects relate to one another;
- the logic which determines which effects occur for each instance of the event;
- the data accesses which are required to support the event processing.

During Logical Design (SSADM Stage 5), ECDs are transformed into Update Process Models. These are Jackson-like structures which can provide a basis for physical process design for certain implementation environments, particularly those which use Jackson Programming techniques.

## Enquiry Process Models

An Enquiry Process Model is a Jackson-like specification of the database processing which is required to satisfy an enquiry.

Enquiry Process Models are developed directly from Enquiry Access Paths and use the standard Jackson notation for sequence, iteration and selection of processing.

For environments using Jackson Structured Programming techniques (or similar), Enquiry Process Models can provide the basis for physical design specifications. Other environments may require different types of specification to be produced (eg structured text) or may find that the EAPs are adequate for physical design purposes.

## Update Process Models

An Update Process Model is a Jackson-like specification of the database processing which is required to satisfy an event handled by the system.

Update Process Models are developed directly from Effect Correspondence Diagrams and use the standard Jackson notation for sequence, iteration and selection of processing.

For environments using Jackson Structured Programming techniques (or similar), Update Process Models can provide the basis for physical design specifications. Other environments may required different types of specification to be produced (eg structured text) or may find that the ECDs are adequate for physical design purposes.

## CONCEPTS AND NOTATION

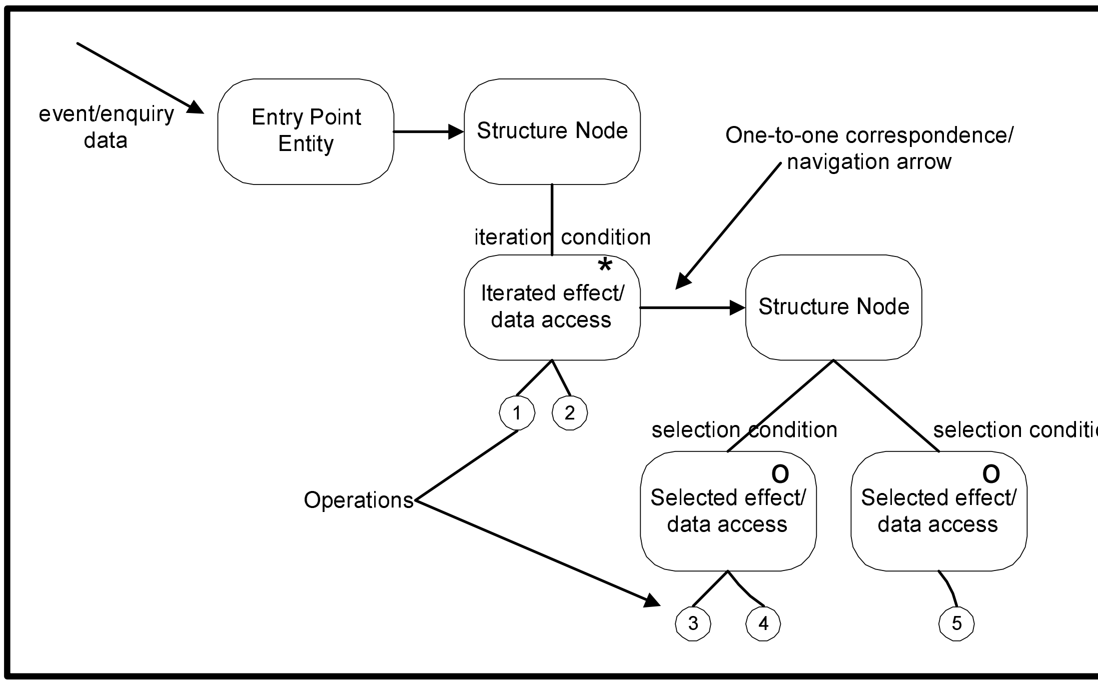
### Enquiry Access Paths/Effect Correspondence Diagrams

Enquiry Access Paths (EAP) and Effect Correspondence Diagrams (ECD) are diagrammatic structures comprising a number of "soft boxes" (ie boxes with rounded corners) which are joined by a series of arrows. Each "soft box" either represents an entity which must be accessed/updated or is a structure node which is used for selection and iteration purposes.

The navigation between different entities is denoted by arrows joining the soft boxes.

The first entity accessed as part of the enquiry/event processing (the "entry point") is denoted by an arrow. The data items which are used to access the entry point entity are placed against this arrow.

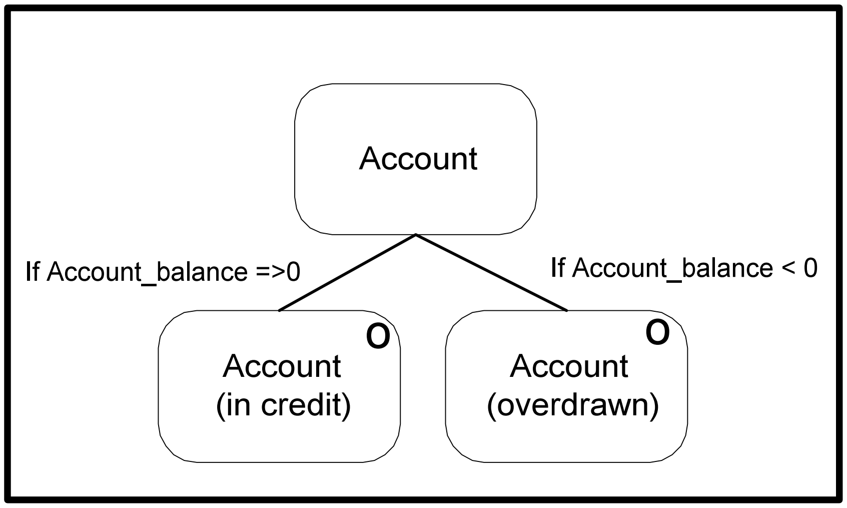
The notation used for EAPs and ECDs is shown in the diagram below:


event/enquiry data
Entry Point Entity
Structure Node
One-to-one correspondence/ navigation arrow
iteration condition
*
Iterated effect/ data access
Structure Node
1
2
selection condition
selection conditi
O
O
Operations
Selected effect/ data access
Selected effect/ data access
3
4
5

## Selections

Selections between different data accesses/updates are represented by a structure node with a box for each of the possible data accesses/updates beneath it. Each box has the symbol "o" in the top right corner.

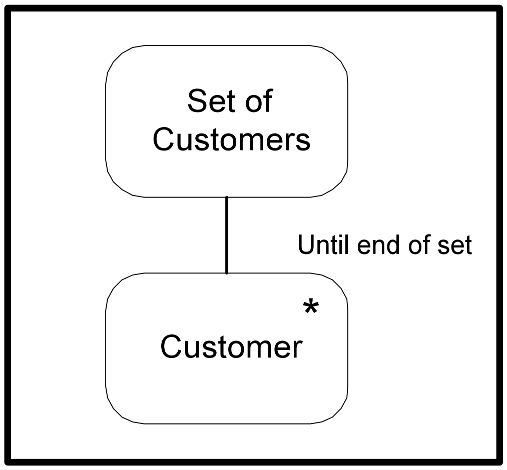
Each selection is annotated with the condition which determines when the data access/update represented by the selection is carried out. An example of a selection structure is as shown below:



## Iterations

A data access/update which is iterated is represented by a structure box with a single box immediately below it. The box contains the symbol "*" in the top right corner. The name of the structure box usually takes the format "Set of <entity name>", where "entity name" refers to the entity for which the data access/update is iterated.

Each iteration may be annotated with the condition which determines the circumstances under which the data access/update is iterated. An example of an iteration structure is shown below.



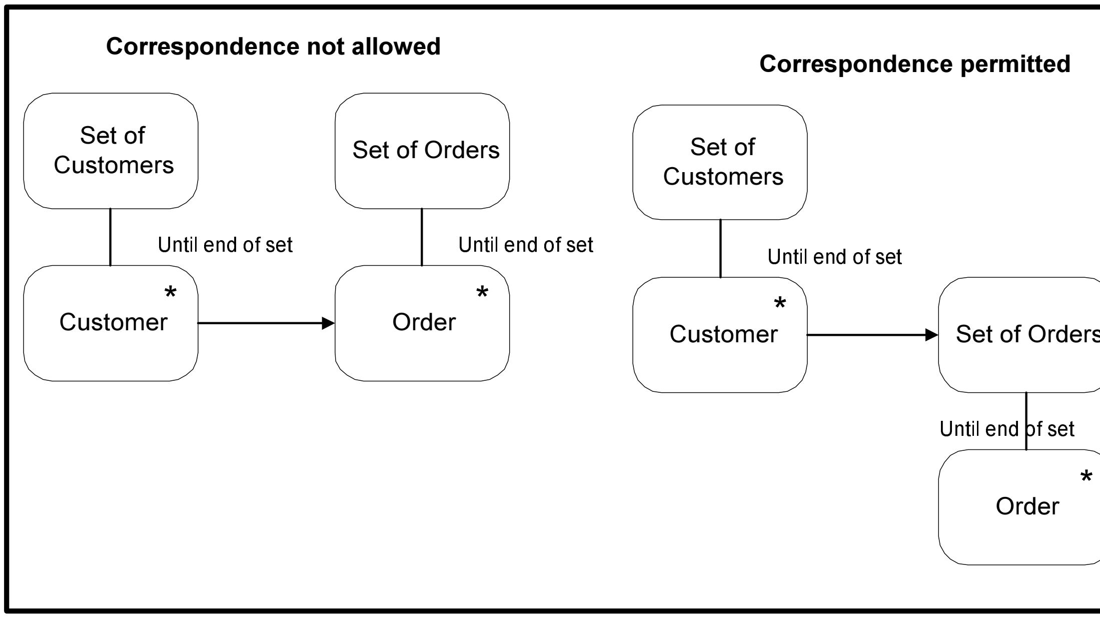
## One-to-One Navigation/Correspondence Arrows

Arrows are used to join two data accesses/updates and to show the direction of navigation through the Required System LDS from the entry point entity. A navigation arrow denotes that on completion of the first data access/update (or on completion of each iteration of a repeating data access/update), the second data access/update must take place.

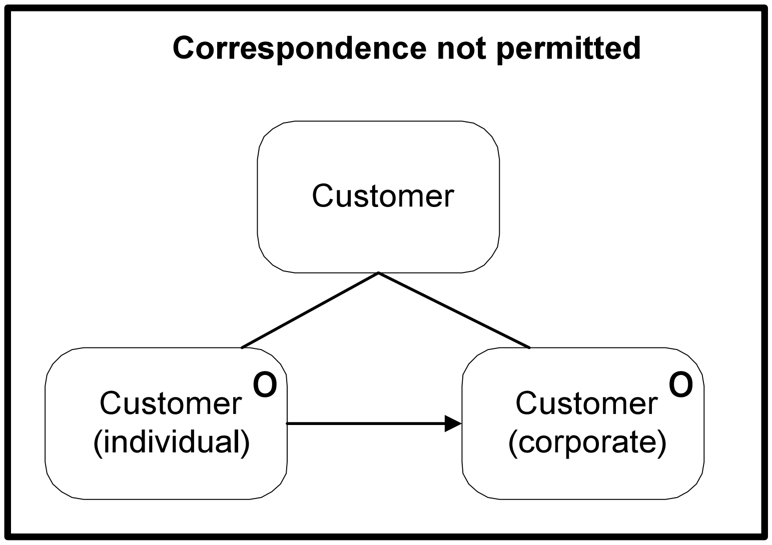
To confuse matters, these arrows are given different names depending on whether they are being drawn on EAPs or ECDs! For EAPs, the arrows are called "One-to-One Navigation Arrows". For ECDs, the arrows are called "Effect Correspondence Arrows". For practical purposes, this difference in names has no impact on the EAPs and ECDs which project teams must produce.

Navigation/Correspondence arrows may not be drawn between the following elements:

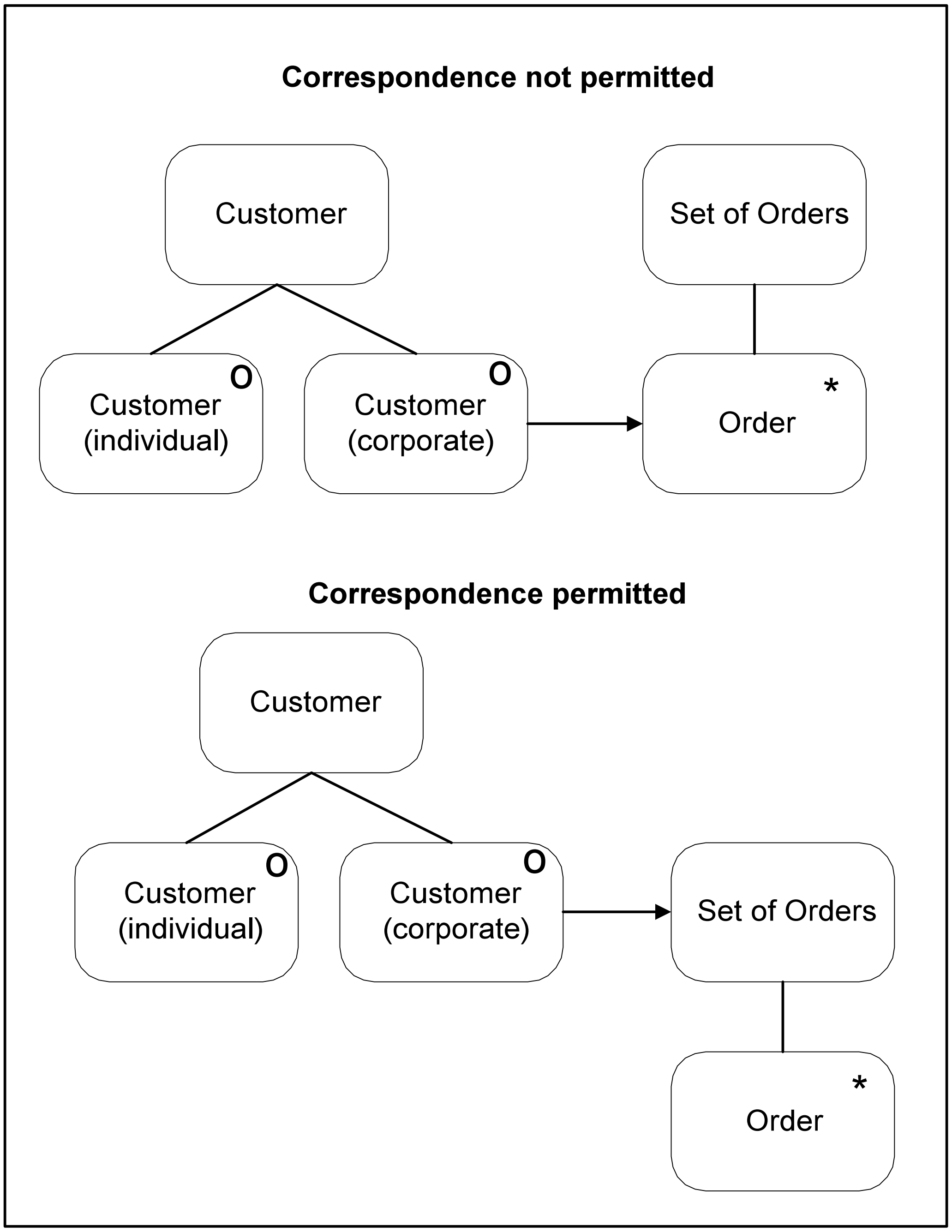
- **between two iterated elements.** NB: However an iterated element may be linked to an iteration structure as in the diagram below:



- **between two selection elements.** The fact that elements are marked as selections means that they are mutually-exclusive of each other. If one selection is chosen, the other data accesses/updates cannot take place.



- **between a selection element and an iterated element.** NB: However a selection element may be linked to an iteration structure (and conversely an iterated element to a selection structure) as shown in the diagram below:



## Entry Points

The data items which are used to access the entry point for each enquiry and event are listed on the EAP/ECD with an arrow against the relevant entity. (All entry points to the LDM will be used during Physical Data Design to ensure that appropriate physical access mechanisms are implemented.)

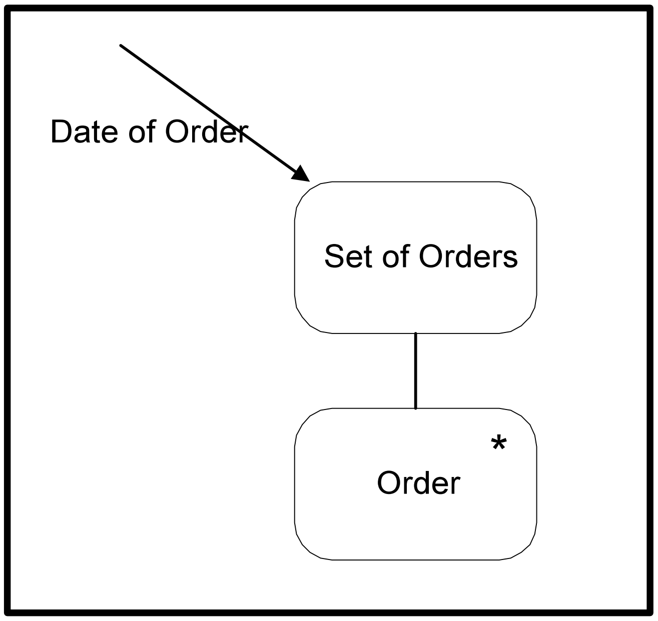
There are three ways in which an entry point entity can be accessed:

- using the primary key of the entity
- using non-key attributes of the entity
- retrieving all occurrences of an entity.

**NB: An entry point entity should not be accessed by any of its foreign keys. In such circumstances, the entry point for the EAP/ECD should be the master entity for which the foreign key is the primary key.**

Where multiple occurrences of an entry point entity are accessed, the access is shown as an iterated structure. If the entity is accessed using non-key attributes, the entry point arrow is annotated with the relevant attributes. If all occurrences of an entity are accessed, the entry point arrow is either left unlabelled or else the label "All occurrences" is placed against it.

An example of an iterated entry point structure is shown below:

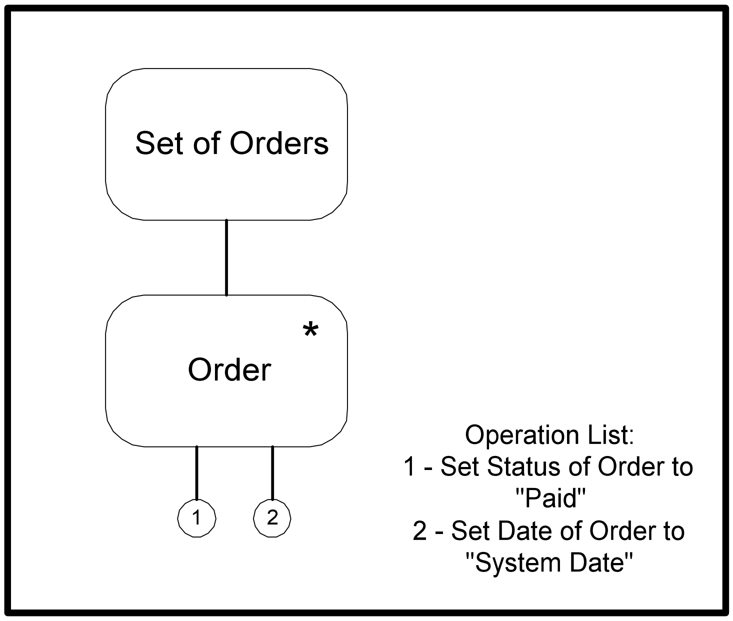


## Operations

Each data access/update on an EAP/ECD comprises one or more operation. An operation consists of a specific data access, data manipulation or data update statement.

Operations can be added to the data access/update boxes on an EAP/ECD as well as to boxes which are introduced as part of iteration and selection structures. Operations are added to a data access, update or structure box in the logical sequence in which they are invoked.

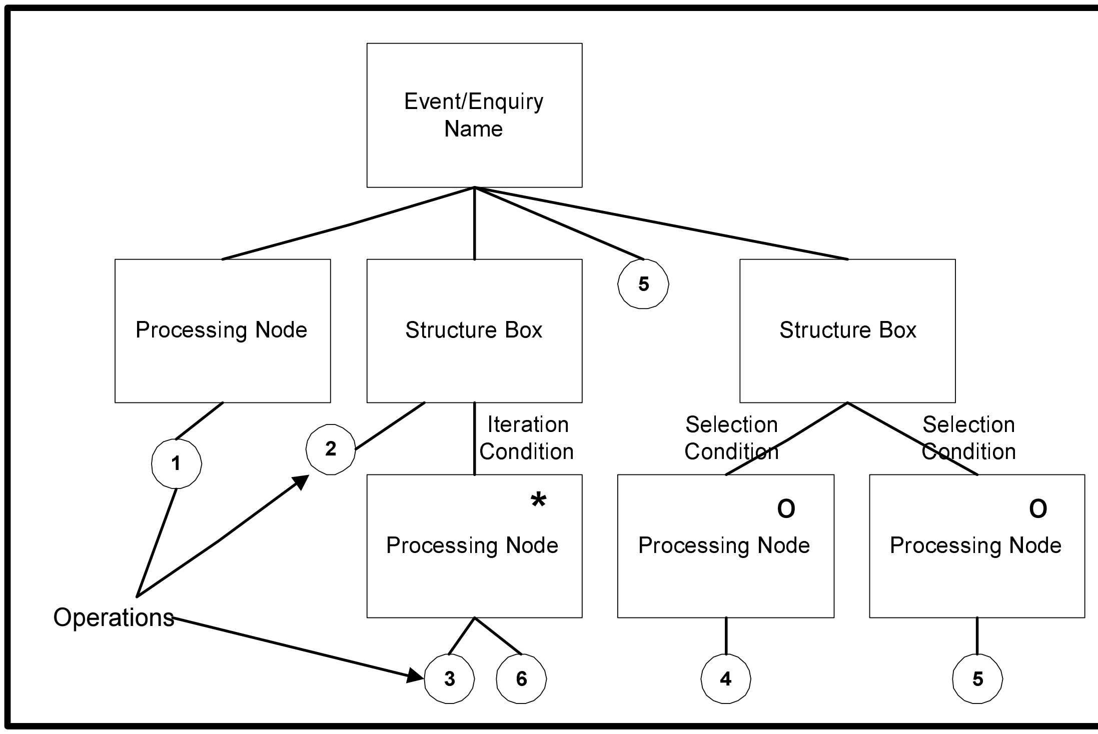
An example of how operations appear on EAPs/ECDs is shown below:


Set of Orders
Order
*
1
2
Operation List:
1 - Set Status of Order to "Paid"
2 - Set Date of Order to "System Date"

Depending on the CASE support available to a project, all operations should be defined centrally (ie once) and referenced wherever used. While SSADM does not dictate a particular syntax for operations, it does provide guidance on the types of operations which should be included on EAPs/ECDs. The actual syntax and operations types to be used should be defined by local organisational/project standards.

## Enquiry/Update Process Models

The notation used for Enquiry and Update Process Models is shown in the diagram below:



Unlike EAPs and ECDs, the models comprise “hard” boxes, ie regular rectangles.

The top box of an Enquiry/Update Process Model contains the name of the event or enquiry being specified.

Each bottom-leaf box must comprise at least one operation. Operations can be added to intermediate structure boxes as well as to the bottom-leaf boxes.

## STEPS

Enquiry Access Paths

Effect Correspondence Diagrams

Enquiry Process Models

Update Process Models

## Enquiry Access Paths

The steps involved in producing an EAP are as follows:

- Define enquiry
- Identify entities to be accessed
- Draw the required view of the Required System LDM
- Redraw the required view as an Enquiry Access Path
- Document the entry point for the enquiry
- Check LDM and add operations and conditions

A worked example demonstrating each of the above steps is provided.

## Define enquiry

An Enquiry Access Path is produced for each enquiry. The name of an Enquiry Access Path is the same as that of the enquiry to which it relates.

The majority of enquiries will be derived from the Requirements Catalogue and Functions, although major enquiries may be derived from the Required System Data Flow Model. Enquiries are documented initially in the Event and Enquiry Catalogue and on the Entity Access Matrix.

Having identified the enquiries to be handled by the system, the data items which are input to and output from the enquiry need to be defined.

## Input data items

All enquiry processes are triggered by the input of data item values. These data item values usually comprise the primary key of an entity (in order to retrieve a specific instance of an entity) or selection criteria (in order to retrieve particular instances of an entity or entities).

The data items which are input to an enquiry process will be identified either:

- as input elements of the I/O Structure relating to the function of which the enquiry is a part

or

- from the Event and Enquiry Catalogue.

**Output data items**

The data items that are output by an enquiry will be identified either:

- as output elements of the I/O Structure relating to the function of which the enquiry is a part

or

- from the Event and Enquiry Catalogue.

**Identify entities to be accessed**

The entities which must be accessed in order to obtain the data for the enquiry's output are identified. If the required data cannot be identified from the LDM or calculated from data on the LDM, it may be that additional entities, relationships or entity attributes need to be added to the LDM.

**Draw the required view of the Required System LDM**

Depending on the CASE support available to the project team, this activity may be carried out in a number of ways, including:

- developing a separate diagram (ie sub-set) of the Logical Data Structure
- annotating the Logical Data Structure.

The Required View is produced as follows:

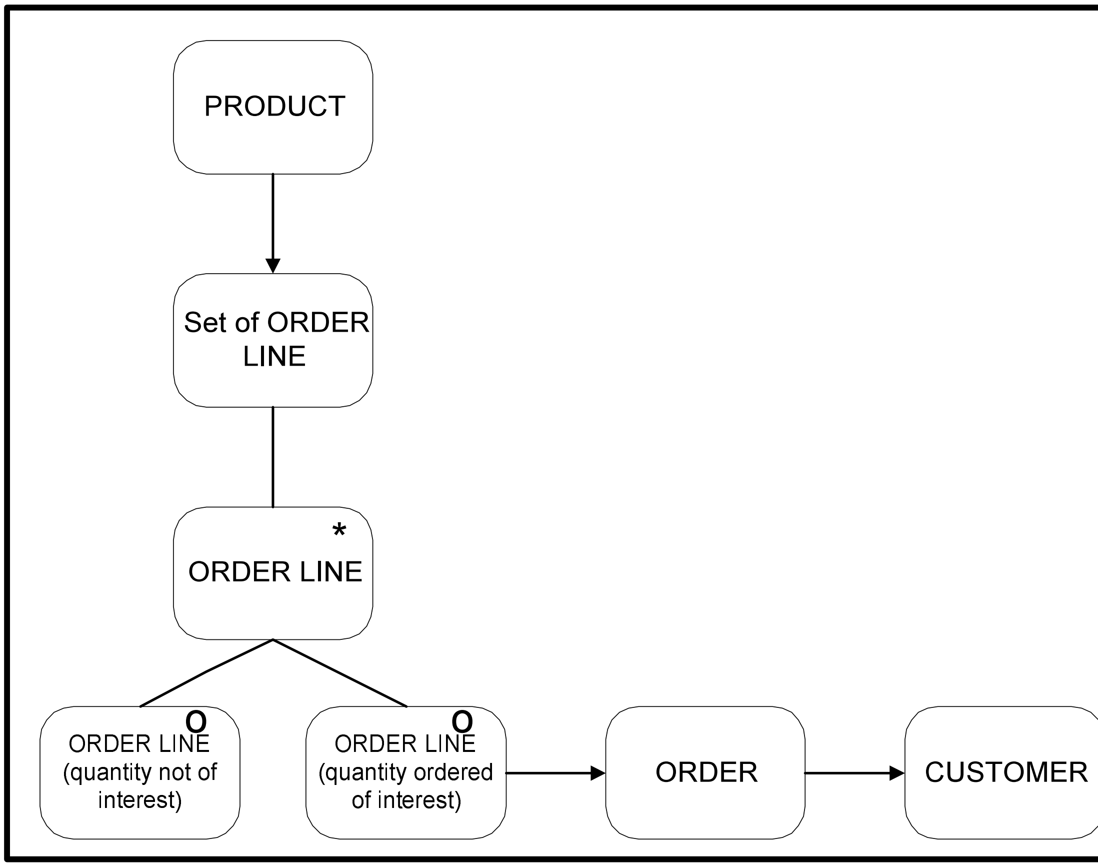
- Identify the entry point entity for the enquiry
- Starting from the entry point entity:
    - position each detail entity which is accessed by going down a relationship (ie from master to detail) beneath the master entity. If there is more than one relationship between the master and detail entities, ensure that the correct relationship is included on the Required View.
    - position the master entity which is accessed by going up a relationship (ie from detail to master) alongside the detail entity. If there is more than one relationship between the master and detail entities, ensure that the correct relationship is included on the Required View.

## Redraw the required view as an Enquiry Access Path

The Required View of the LDM is redrawn as an Enquiry Access Path. Guidelines for generating the EAP are as follows:

- each entity on the Required View should appear on the EAP.
- each relationship line on the Required View becomes a one-to-one navigation arrow. The navigation arrows show the logical order in which the entities are accessed.
- for each entity which is accessed by travelling down a relationship (ie from master to detail), an iteration structure must be introduced and the condition controlling the iteration defined.
- for each entity which is accessed by travelling down a relationship (ie from master to detail), the analyst needs to consider what happens with each occurrence which is accessed. If each occurrence of the detail entity accessed is handled in the same way, the iteration structure remains unaltered. However, if occurrences of the entity are treated in a different way depending on the values of certain attributes, a selection structure must be included to reflect the different processing which takes places and the condition governing the selection defined.

For example, if an enquiry was only interested in retrieving orders where more than a certain quantity of a product was ordered, the EAP would be as follows:


PRODUCT
Set of ORDER LINE
ORDER LINE *
ORDER LINE (quantity not of interest) O
ORDER LINE (quantity ordered of interest) O
ORDER
CUSTOMER

The above structure shows that each instance of an order line is either ignored if the value for attribute 'Quantity Ordered' is less than a certain amount or else details of the order are retrieved.

Add further selection structures where necessary to handle the following:

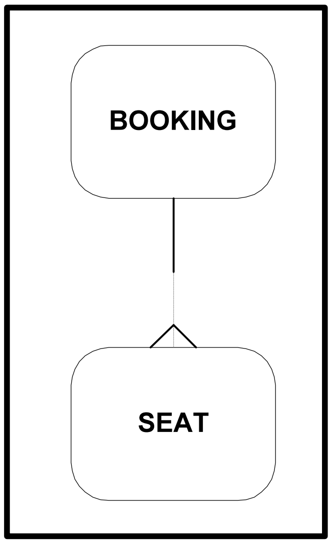
- **Optional masters**
- **Processing of master entity varies according to attribute values**

## Optional masters

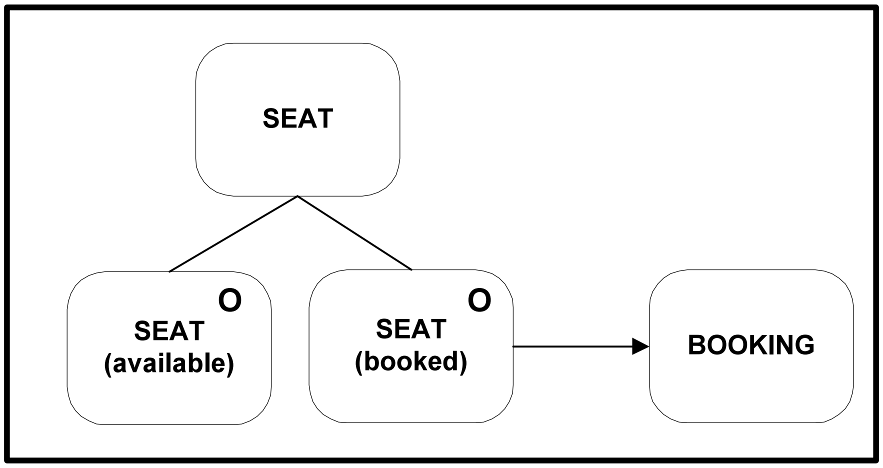
For each entity which is accessed by travelling up a relationship (ie from detail to master), examine whether the relationship is optional from the detail entity’s perspective.

If the relationship is optional, a selection structure is included to reflect the different processing which occurs depending on whether the detail entity is linked to an occurrence of the master.

For example, assume the following LDM which handles a booking system for a football stadium.



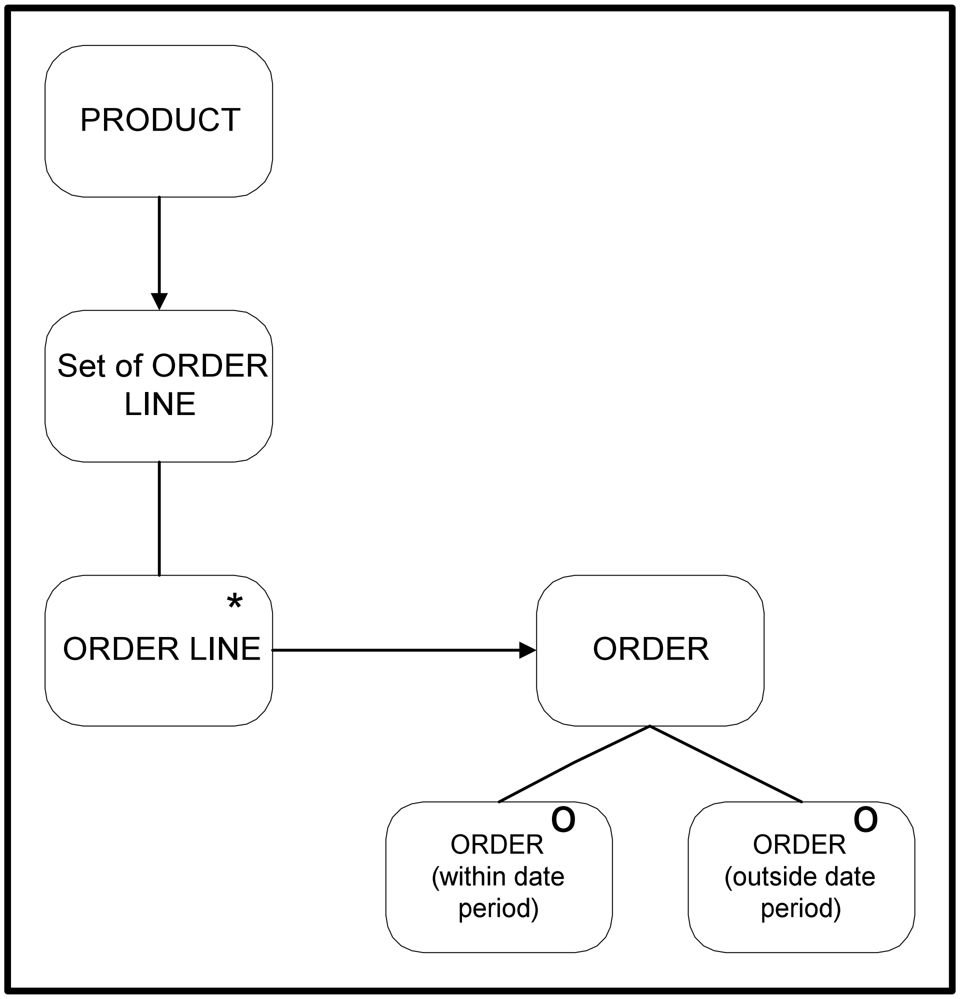
The relationship is optional from the SEAT entity's perspective as not all seats may be sold. If an enquiry had to retrieve the booking details for a seat, the EAP would look as follows:



## Processing of master entity varies according to attribute values

For each entity which is accessed by travelling up a relationship (ie from detail to master), consider what happens to the master occurrence which is accessed. If the master entity is processed in a particular way depending on the value of certain of its attributes, a selection structure is included to reflect the different processing which takes place and the conditions governing the selection defined.

For example, if an enquiry was only interested in orders which were placed for a certain product within a certain date range, the EAP would look as follows:



**Document the entry point for the enquiry**

The attributes which are used to access the entry point entity are listed on the EAP with an arrow against the relevant entity.

If multiple occurrences of the entry point entity are accessed, an iteration structure needs to be added to the EAP along with the condition which governs the iterated data accesses.

An entry point entity should not be accessed by any of its foreign keys. In such circumstances, the entry point should be the master entity for which the foreign key is the primary key.

Should an entry point be several entities and relationships away from the entities required for output purposes, then all these intermediate entities and relationships must be included on the EAP.

**Check LDM and add operations and conditions**

The EAP needs to be checked to ensure that all the required data can be obtained.

This is done by checking that each of the entities included on the EAP can be accessed given the navigation routes which have been modelled. Starting with the entry point entity, each access on the EAP must be capable of being implemented using one of the following logical constructs:

- Read <Entry point entity> using <enquiry trigger attributes> to obtain <attributes>
- Read <Entity> using <primary key attributes> to obtain <attributes>
- Read all occurrences of <Entry point entity> to obtain <attributes>
- Read <Master entity> from occurrence of <Detail entity> to obtain <attributes>
- Read next occurrence of <Detail entity> from <Master entity> to obtain <attributes>

**NB: The phrase "to obtain <attributes>" is ignored if the entity is being retrieved for navigation purposes only and no attributes from the entity are being retrieved for output purposes.**

If the above read operations are not sufficient to obtain all the data required for output purposes or the data obtained is not in the format required for output purposes, then either:

- the LDM (and resultant EAPs) need to be modified

or

- a processing solution needs to be identified and included either on the EAP as operations or as part of the relevant function definition. For example, if there is a requirement to sort/calculate data for output purposes, appropriate operations can be added to the EAP or included as part of the relevant function description.

The EAP is completed by adding:

- the data access and data manipulation operations.
- the iteration and selection conditions.

## EAP Worked Example

For the purposes of demonstrating how an EAP is produced, the following enquiry will be used:

**Enquiry: Produce Product History List**

**Description:** Produce a list of customers who have ordered a particular product.

**Input Data Items:**

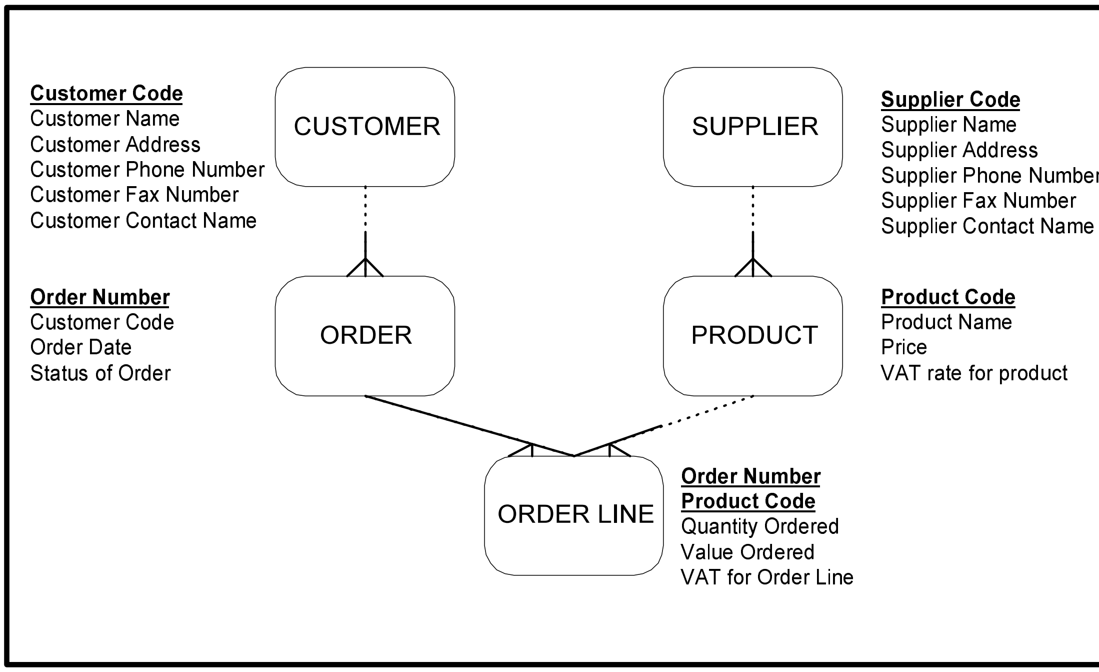
Product Code

**Output Data Items:**

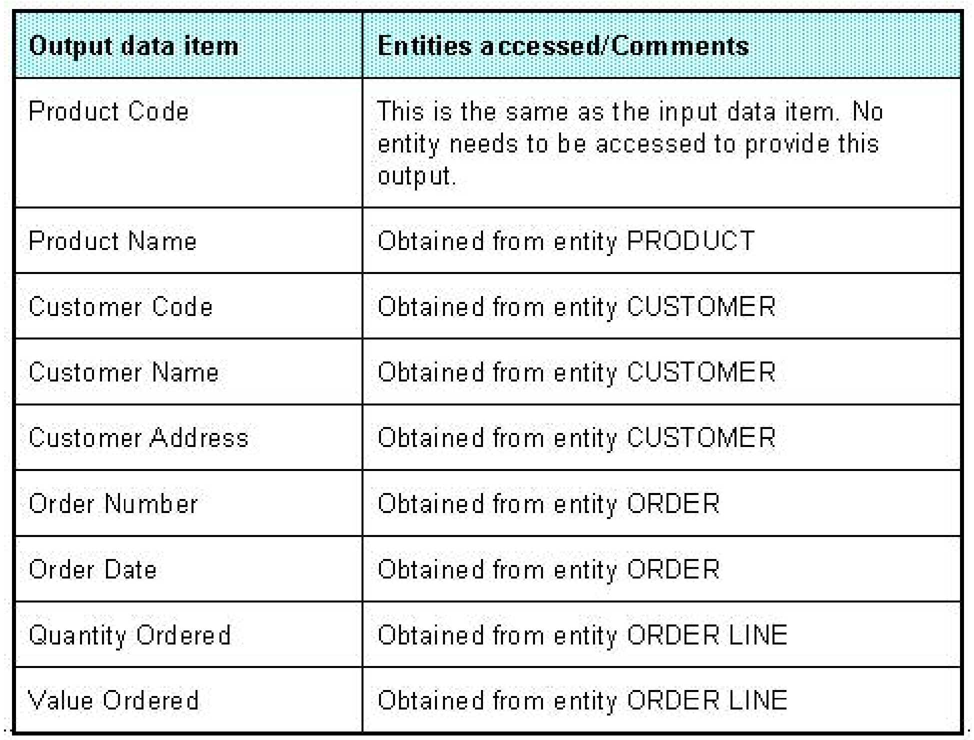
Product Code
Product Name
Customer Code
Customer Name
Customer Address
Order Number
Order Date
Quantity Ordered
Value Ordered

## EAP Worked Example: Step 1 – Identify entities to be accessed

The LDM for the worked example is as follows:



The data items which must be output by the enquiry are as follows:



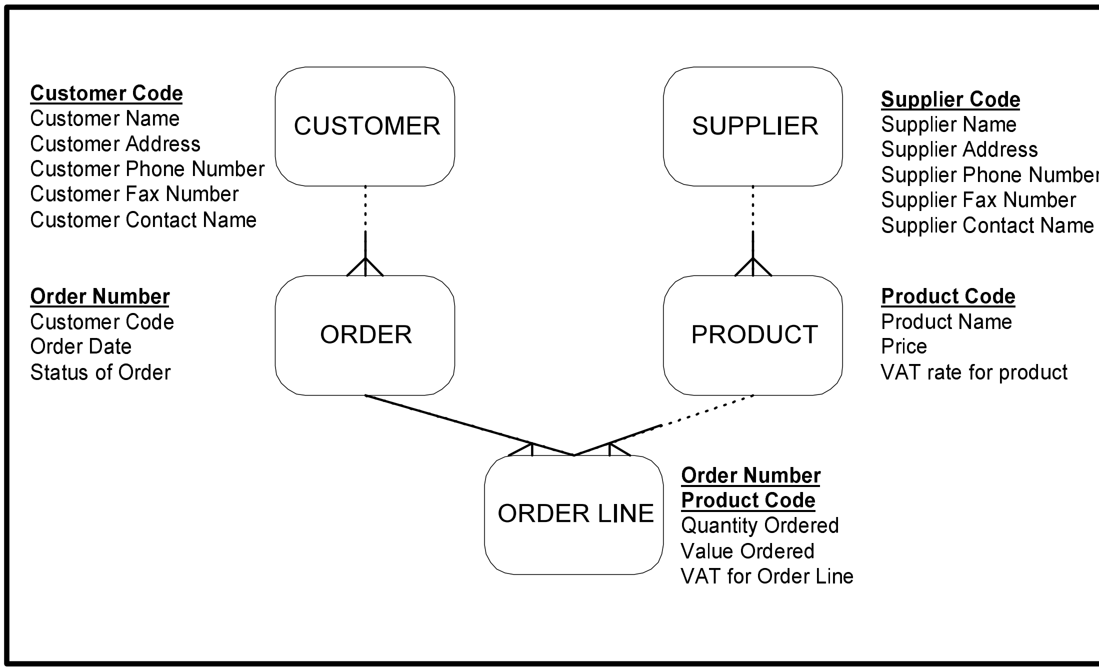
| Output data item | Entities accessed/Comments |
|---|---|
| Product Code | This is the same as the input data item. No entity needs to be accessed to provide this output. |
| Product Name | Obtained from entity PRODUCT |
| Customer Code | Obtained from entity CUSTOMER |
| Customer Name | Obtained from entity CUSTOMER |
| Customer Address | Obtained from entity CUSTOMER |
| Order Number | Obtained from entity ORDER |
| Order Date | Obtained from entity ORDER |
| Quantity Ordered | Obtained from entity ORDER LINE |
| Value Ordered | Obtained from entity ORDER LINE |

The entities which need to be accessed for the purposes of this enquiry are:

PRODUCT
CUSTOMER
ORDER
ORDER LINE

## EAP Worked Example: Step 2, Draw the required view of the Required System LDM

The LDM for the worked example is as follows:



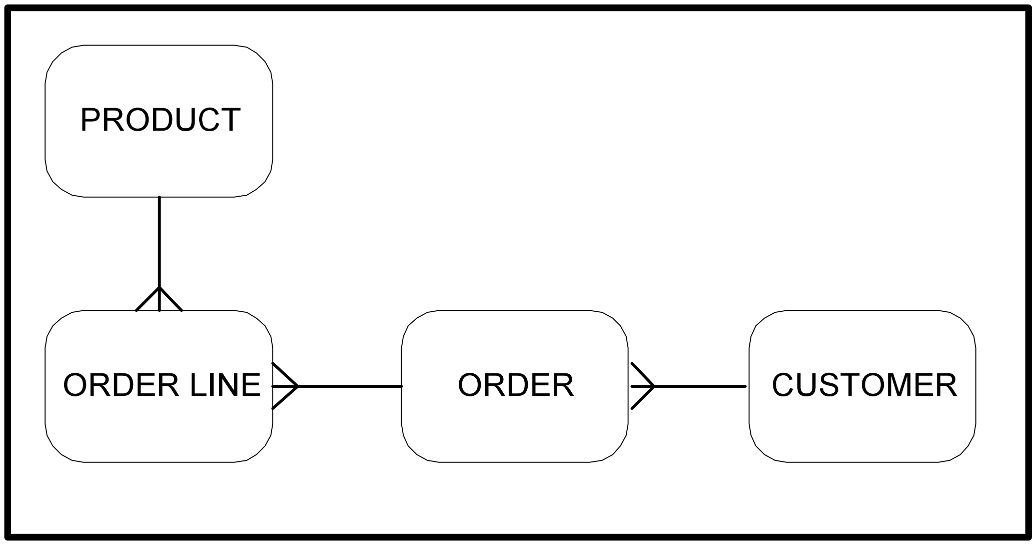
For the worked example, the entry point to the LDM is entity PRODUCT.

From PRODUCT, the detail entity ORDER LINE is accessed to identify those instances where the product was ordered and to retrieve attributes 'Quantity Ordered' and 'Value Ordered'.

For each occurrence of ORDER LINE accessed, the master entity ORDER is accessed to retrieve attributes 'Order Number' and 'Order Date'.

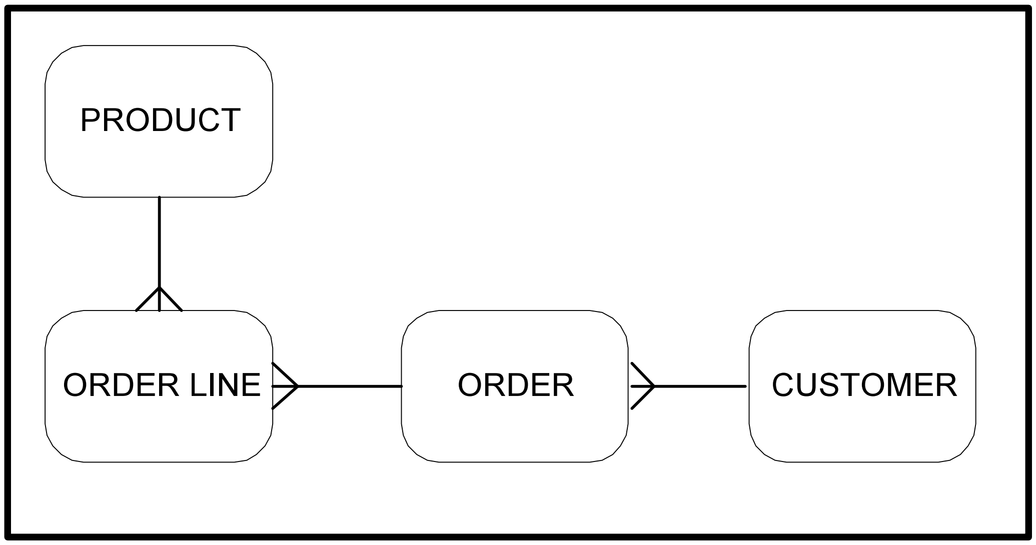
Finally, for each occurrence of entity ORDER accessed, the master entity CUSTOMER is accessed to retrieve 'Customer Code', 'Customer Name' and 'Customer Address'.

The Required View of the LDM for the worked example is as follows:

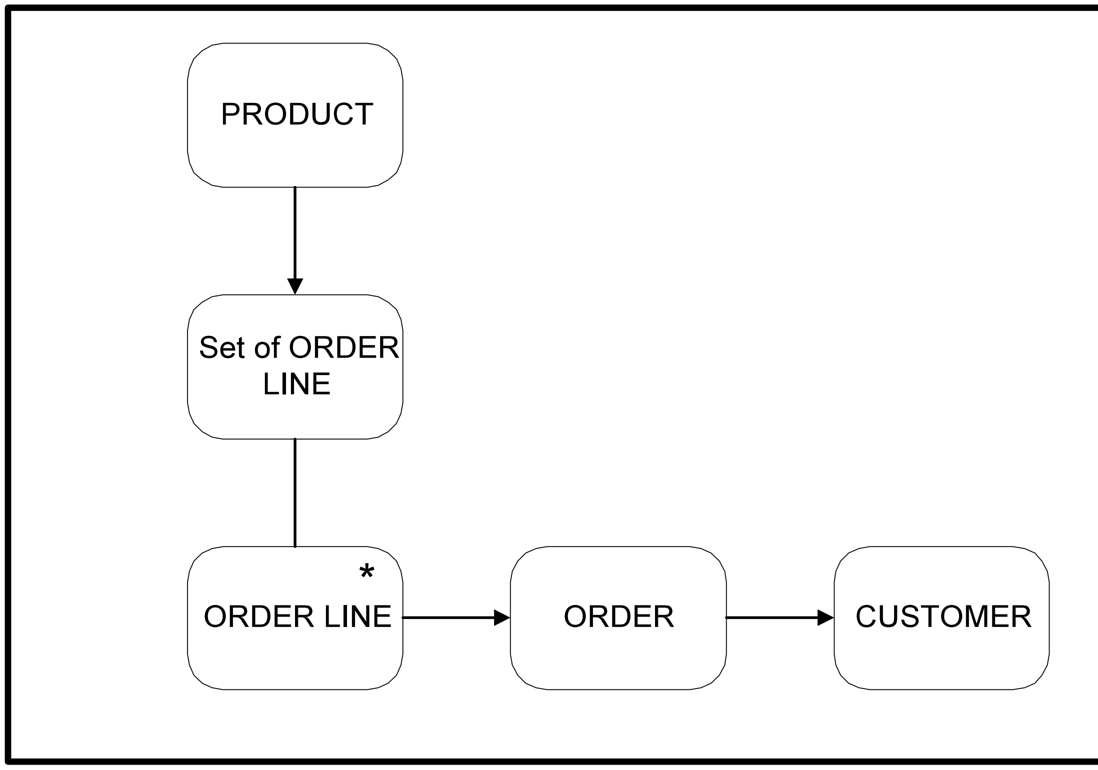


**EAP Worked Example: Step 3, Redraw the required view as an Enquiry Access Path**

The required view for our worked example is as follows:

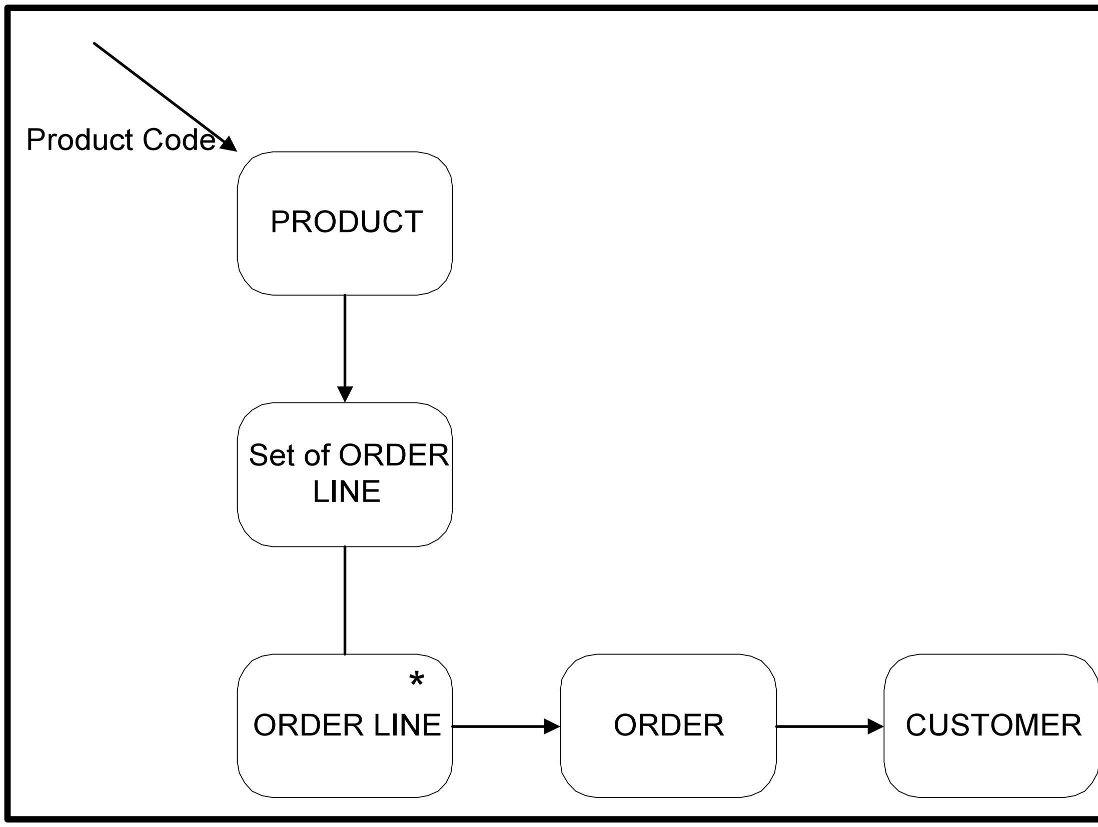


This is transformed into the following EAP:



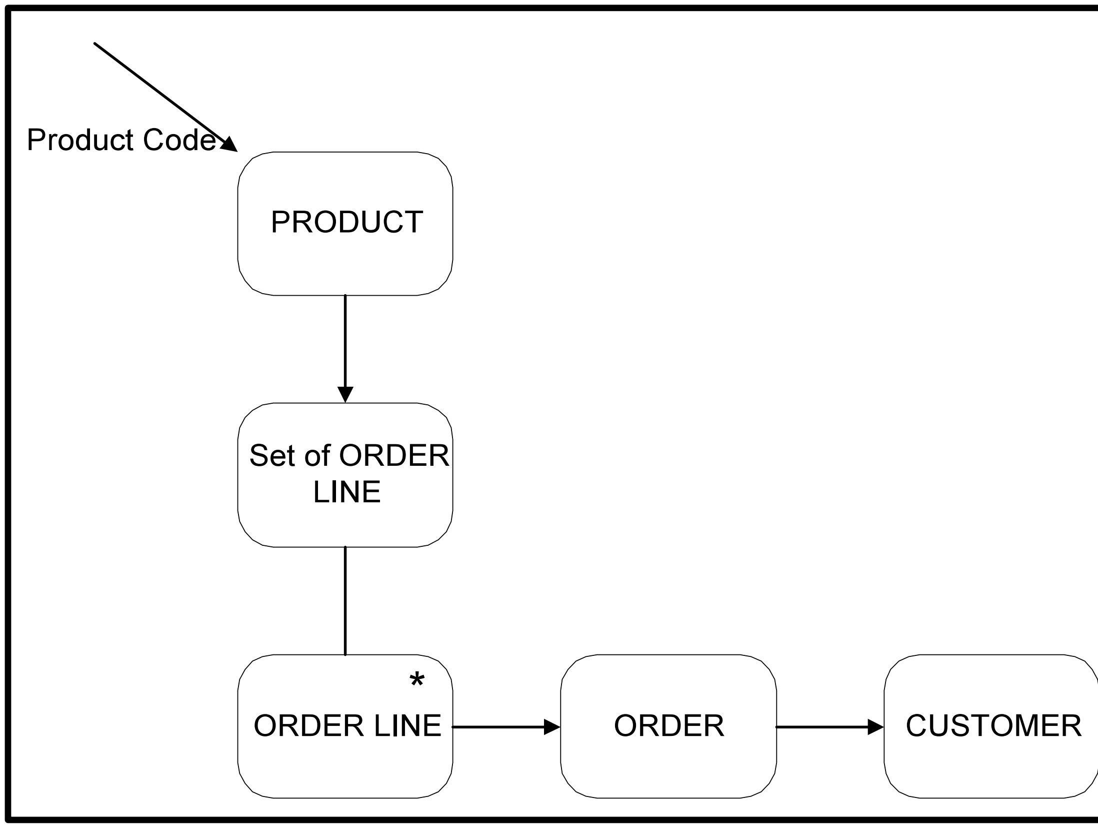
**EAP Worked Example: Step 4, Document the entry point for the enquiry**

The following diagram shows how the EAP has been enhanced to incorporate the entry point:



## EAP Worked Example: Step 5 - Check LDM and add operations and conditions

The EAP for the worked example is as follows:

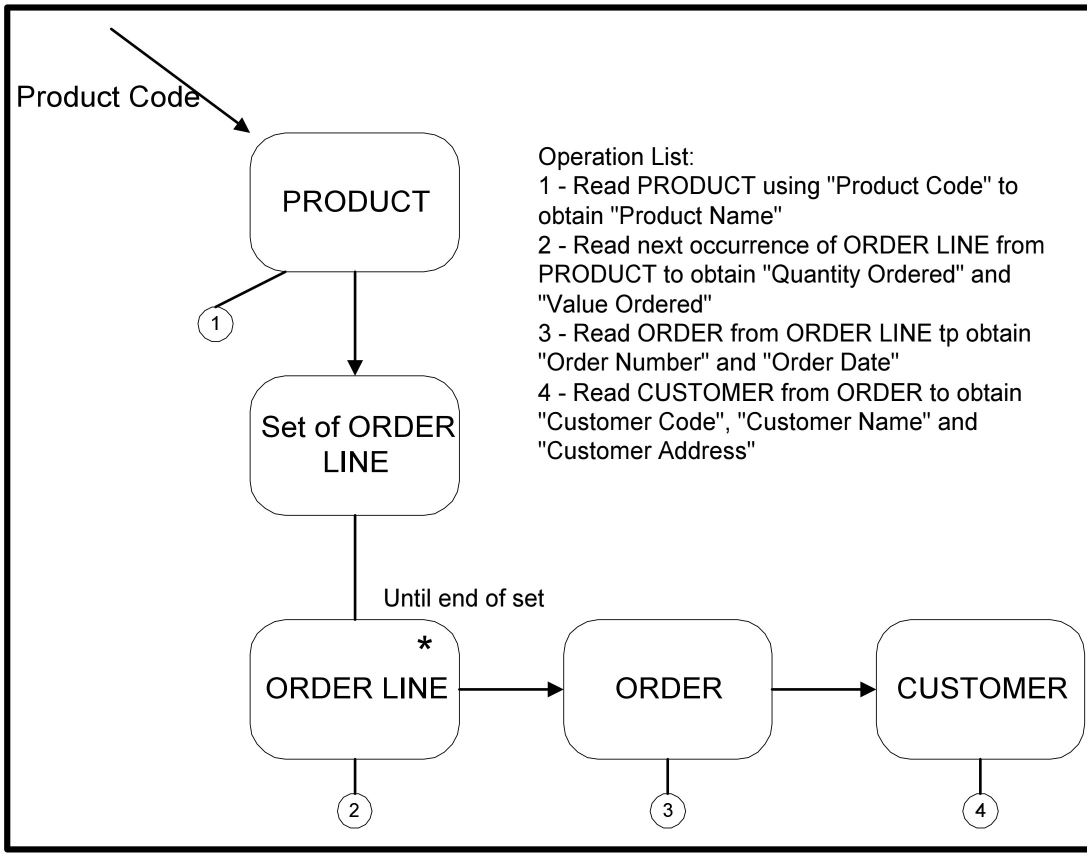


The EAP can be read as follows:

- Read PRODUCT using 'Product Code' to obtain 'Product Name'
- Read next occurrence of ORDER LINE from PRODUCT to obtain 'Quantity Ordered' and 'Value Ordered'
- Read ORDER from ORDER LINE to obtain 'Order Number' and 'Order Date'
- Read CUSTOMER from ORDER to obtain 'Customer Code', 'Customer Name' and 'Customer Address'.

All the data which must be accessed by the enquiry can be retrieved.

To complete the EAP, operations and conditions are added as follows:


Product Code
PRODUCT
Set of ORDER LINE
Until end of set
*
ORDER LINE
ORDER
CUSTOMER
1
2
3
4
Operation List:
1 - Read PRODUCT using "Product Code" to obtain "Product Name"
2 - Read next occurrence of ORDER LINE from PRODUCT to obtain "Quantity Ordered" and "Value Ordered"
3 - Read ORDER from ORDER LINE tp obtain "Order Number" and "Order Date"
4 - Read CUSTOMER from ORDER to obtain "Customer Code", "Customer Name" and "Customer Address"

**Effect Correspondence Diagrams**

The steps involved in producing an ECD are as follows:

- **Understand Event requirements**
- **Draw a box for each entity affected by the event**
- **Add a box for each entity role**
- **Add all selections**
- **Add iteration structures**
- **Identify entry point for event**
- **Define correspondences**
- **Add conditions to selections and iterations**
- **Add operations**

- **Validate ECD**

A worked example demonstrating each of the above steps is provided.

## Understand Event requirements

Before starting to build the ECD for an event, it is important to have a good understanding of what the purpose of the event is and the processing which is invoked in response to the event.

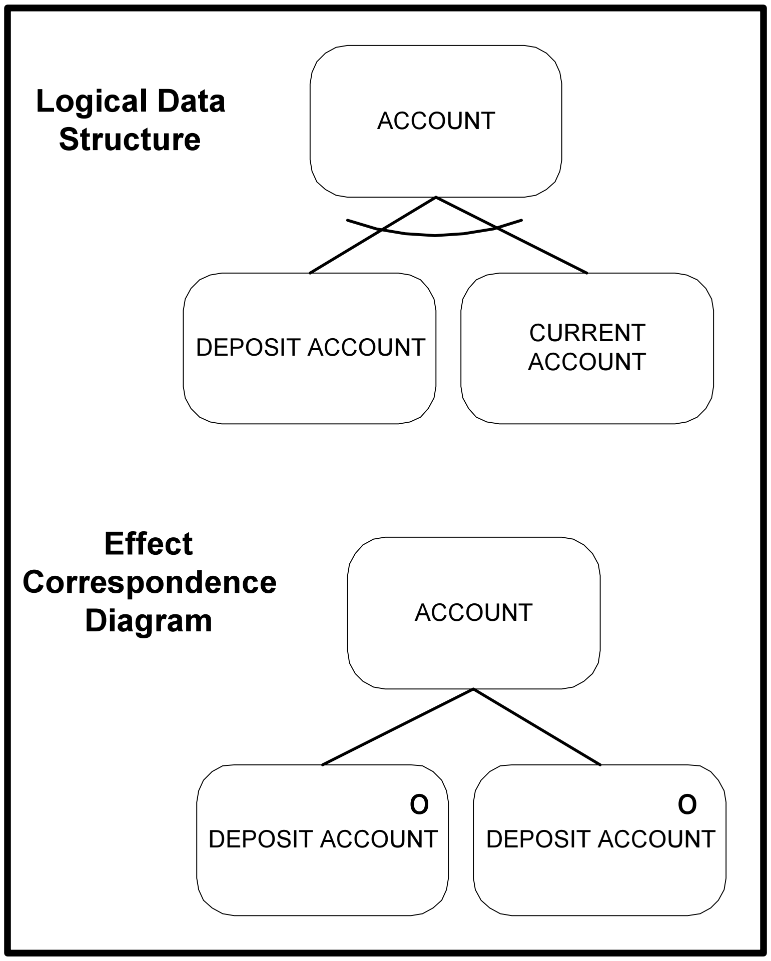
The Event and Enquiry Catalogue will provide an overview of the event. The Function Definition of which the event forms a part will set the event in the context of the overall User process. The Entity Access Matrix will identify the entities which are updated by the event and the nature of each update.

## Draw a box for each entity affected by the event

By examining the Entity Access Matrix, identify each entity which is updated by the relevant event and draw a box for each entity on the emerging ECD.

Entities which are shown on the Entity Access Matrix as being updated by the event **solely** in terms of a Gain or Lose effect must be carefully considered. If these Gain and Lose effects are included on the ELHs for the entities, then these entities must be included on the ECD. If these effects are not shown on the relevant ELHs, the entities should be excluded from the ECD.

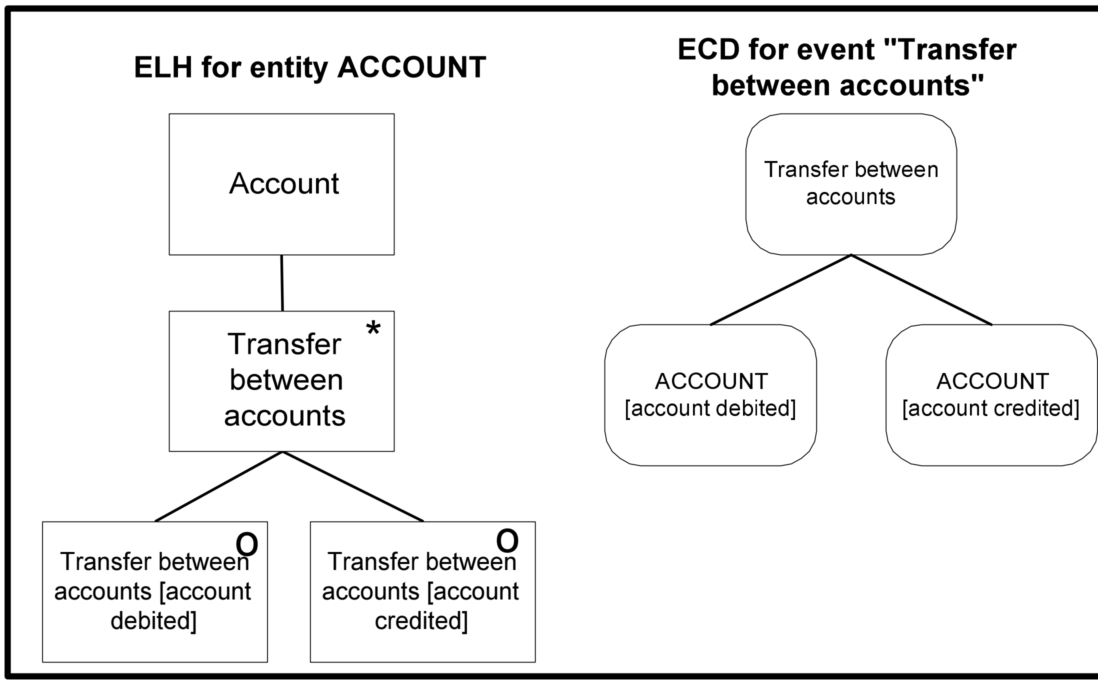
For the purposes of ECDs, entity sub and super-types should be shown separately. A box is created for the super-type and a selection structure is created for each of the sub-types. The following example shows how an ECD updating an entity super/sub-type structure would appear:



**Add a box for each entity role**

ELHs will show if an entity assumes two or more roles for an event. Each of these roles must be reflected on the ECD as separate processing will have to be specified for each role.

In the example below, the ELH for the entity ACCOUNT shows that when money is transferred between accounts, two instances of entity ACCOUNT are affected, namely the account from which money is transferred and the account to which money is transferred. These two entity roles are included on the ECD for event “Transfer between accounts”.

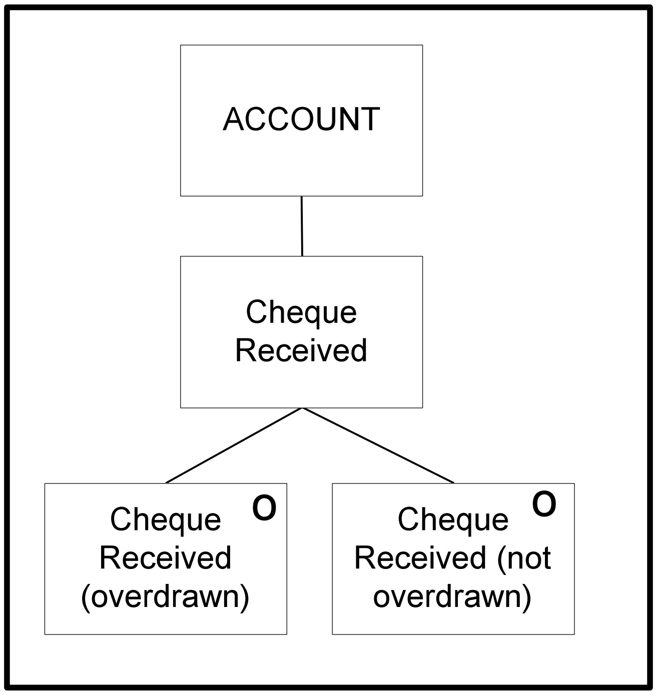

ELH for entity ACCOUNT
Account
Transfer between accounts
*
Transfer between accounts [account debited]
o
Transfer between accounts [account credited]
o
ECD for event "Transfer between accounts"
Transfer between accounts
ACCOUNT [account debited]
ACCOUNT [account credited]

## Add all selections

Where an event affects an entity (or entity role) in two or more mutually exclusive ways, draw a box for each mutually exclusive effect below the relevant entity or entity role box and make each of these boxes a selection.

Mutually exclusive effects are shown on ELHs by the use of an effect qualifier after the event name. These effects must be shown on the ECD as separate processing will have to be specified for each effect.

In the example, below, the ELH for entity ACCOUNT shows that the event *Cheque Received* has two mutually exclusive effects on the entity depending on whether the account is overdrawn.



## Add iteration structures

If more than one occurrence of any of the entities (or entity roles) on the ECD can be affected by the event, identify these entities (or roles) by:

- adding an iteration structure box above the entity (or entity role box)

and

- making the entity (or entity role) box iterate.

Iterative effects can be identified by examining the relationships on the Required System LDS. Where a Master to Detail relationship exists on the LDS and there is a need to update more than one occurrence of the Detail for a specific occurrence of the Master, an iteration structure must be included for the Detail entity.

## Identify entry point for event

The attributes which are used to access the entry point entity are listed on the ECD with an arrow against the relevant entity.

If multiple occurrences of the entry point entity are accessed, an iteration structure needs to be added to the ECD along with the condition which governs the iterated data accesses/updates.

An entry point entity should not be accessed by any of its foreign keys. In such circumstances, the entry point should be the master entity for which the foreign key is the primary key.

**Should an entry point be several entities and relationships away from the entities required for output purposes, then all these intermediate entities and relationships must be included on the ECD.**

## Define correspondences

Having ensured that all the effects for an event have been taken from the relevant ELHs and included on the ECD, the correspondences between the effects must now be specified.

Starting from the entry point entity, the LDM is navigated to identify the logical sequence in which the effects shown on the ECD will take place. By navigating the LDM, it is likely that data accesses (and possibly updates) may be identified as being missing from the ECD. All missing updates and accesses must be included on the ECD and also added to the Entity Access Matrix.

## One-to-one correspondence

A one-to-one correspondence between effects usually occurs where the entity (or entity role) under consideration is accessed from an occurrence of one of its detail entities, ie the LDM is being traversed from detail to master.

One-to-one correspondence is identified by taking each entity (or entity role) in turn and asking whether the specific entity occurrence has been accessed via one its detail entities. If the answer is “Yes”, a one-to-one correspondence has been identified and the master and detail entities are joined by a correspondence arrow, with the arrow head pointing towards the master entity.

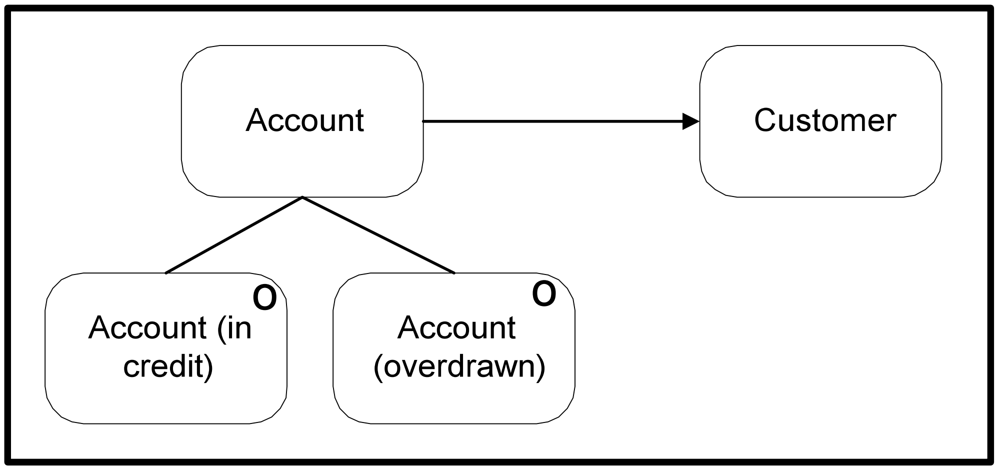
**NB: One-to-one correspondences can also occur when it is possible to navigate from one entity directly to another entity even if the two entity types are not related to one another on the LDM. This may occur where the primary key of the second entity has either been supplied as part of the input data for the event or it can be derived/obtained from data in entities already accessed.**

When identifying one-to-one correspondences, particular attention must be paid to mutually exclusive effects.

## Mutually exclusive effects

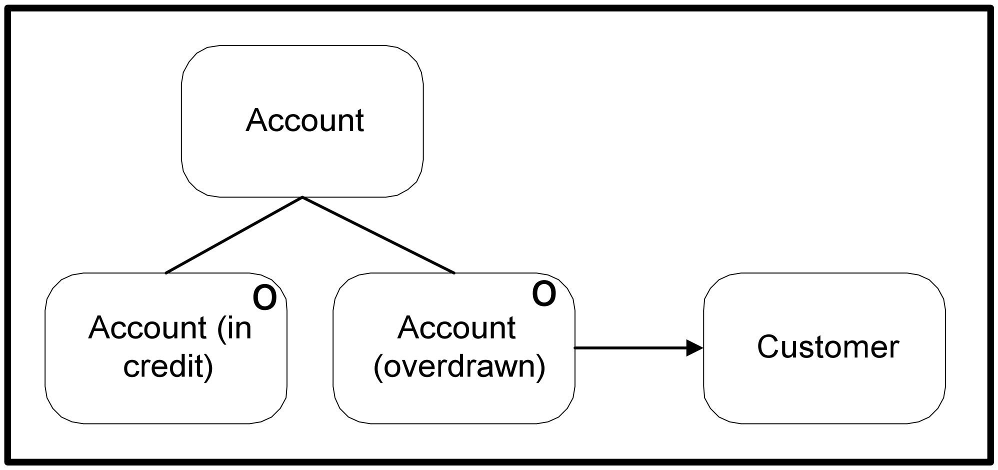
If there are two or more mutually exclusive effects for a detail entity, the analyst must consider whether the effect on the master entity will be the same regardless of which mutually exclusive effect takes place. If the master entity is updated in the same manner irrespective of the mutually exclusive effect which takes place, a one-to-one correspondence is shown between the master entity and the selection structure for the mutually exclusive effects.

In the example below, entity CUSTOMER is updated regardless of which of the two mutually exclusive effects on entity ACCOUNT takes place. Correspondence is shown between entity CUSTOMER and the selection structure.



If however, the effect on the master entity only occurs depending on which of the mutually exclusive effects which takes place, one-to-one correspondence is shown between the master entity and the relevant mutually exclusive effect.

In the example below, entity CUSTOMER is only updated if effect *Account (Overdrawn)* occurs. Correspondence is therefore shown between entity CUSTOMER and this effect:



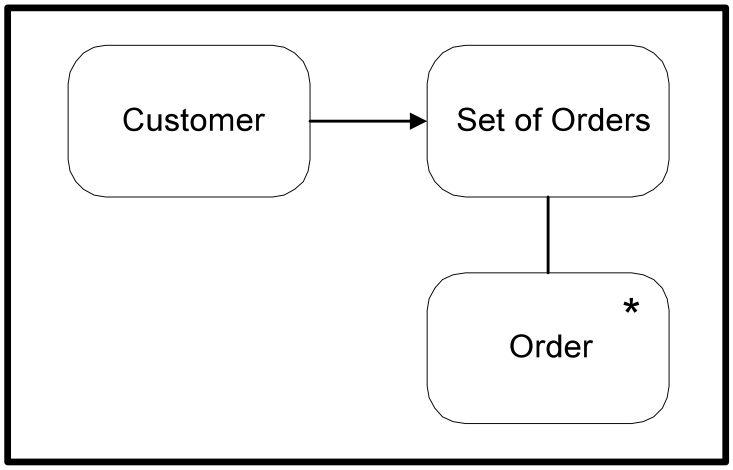
## Correspondence with iterated effects

Where a One to Many relationship exists on the LDM and the processing for an ECD involves going from master to detail, the correspondence between the master and detail entities needs to be carefully considered.

If only a single occurrence of the detail entity has to be accessed, then a correspondence should be drawn between the master and a single instance of the detail entity. (This situation can arise either because the key of the detail has been provided as part of the event data or there is a condition which will ensure that only one occurrence of the detail is selected, eg to retrieve the most recent occurrence of the detail.) As part of Step ‘Add Iteration Structures’, an iterated structure may have been introduced for the detail entity. In such cases, the iterated structure must be removed.

If more than one occurrence of the detail is to be accessed, an iteration structure is introduced for the effect on the detail entity. A one-to-one correspondence is then drawn between the master entity and the iterated structure.

In the example below, multiple occurrences of ORDER are updated for a given occurrence of CUSTOMER:



## Add conditions to selections and iterations

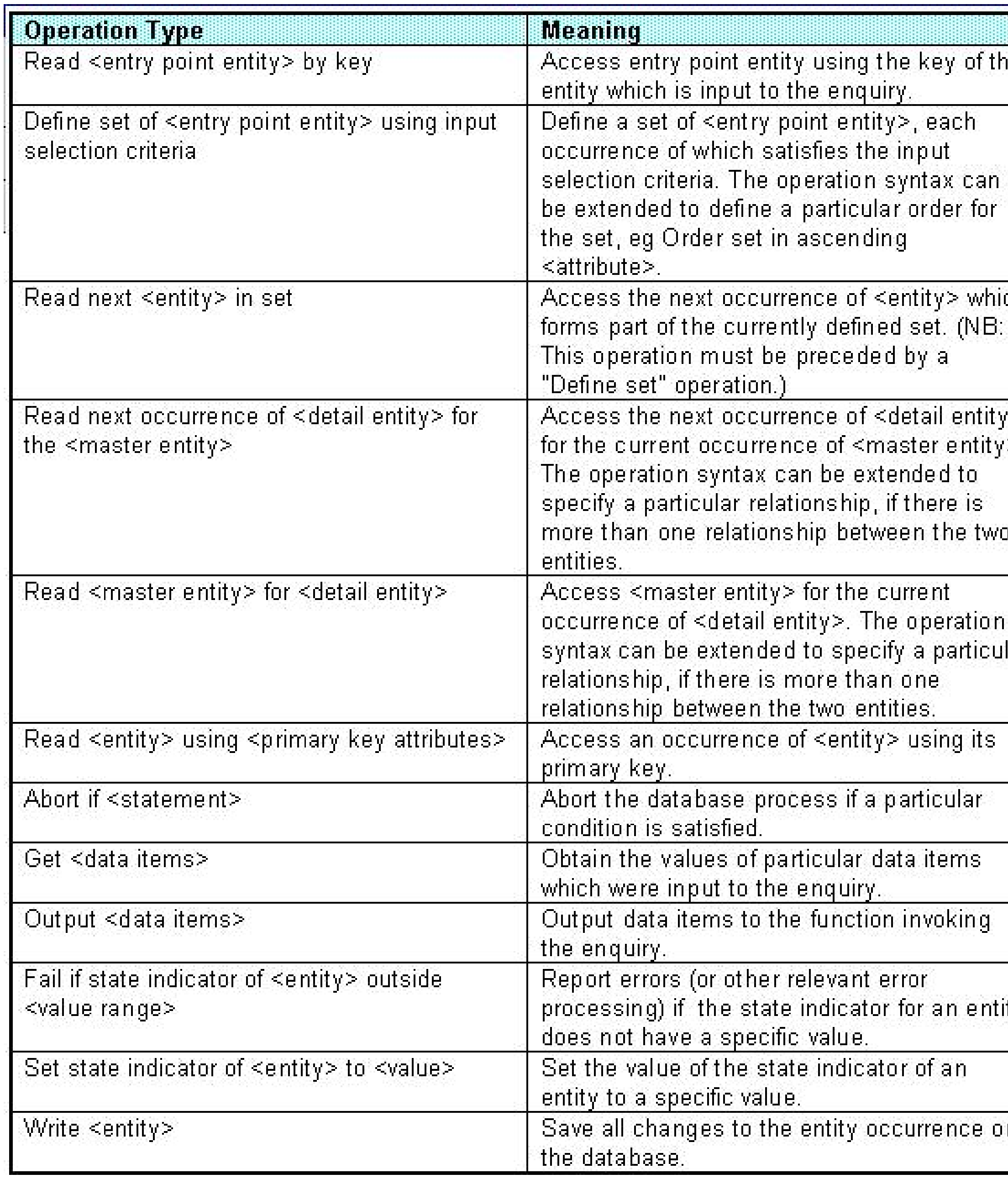
The ECD structure must be enhanced by adding conditions to each option of a selection and to each iteration. The condition must define the circumstances under which either a particular selection element will be chosen or an iteration will be executed.

Selection conditions often test an entity’s current state indicator value. Iteration conditions often test for “end of set of entity occurrences”.

## Add operations

Each effect which is taken from an ELH will already have been defined in terms of ELH operations. These operations must be copied from the ELH and included as part of the effect on the ECD.

In addition to the operations taken from the ELHs, the following operation types must be added where appropriate:



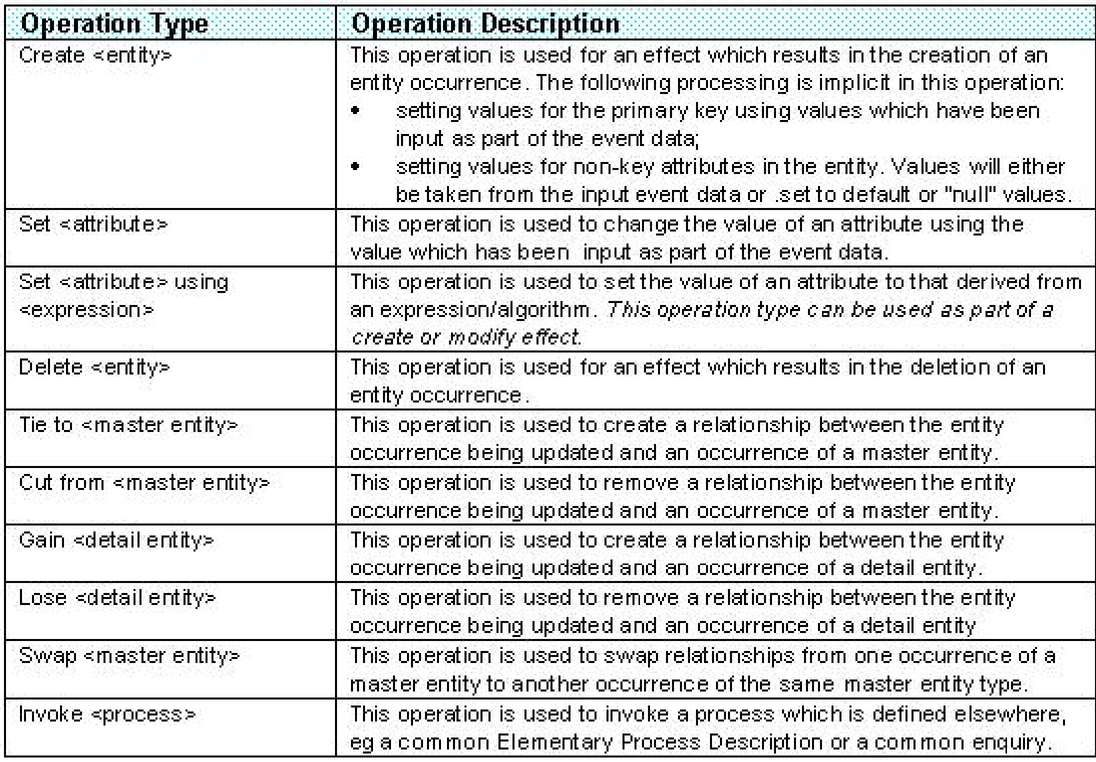
| Operation Type | Meaning |
|---|---|
| Read <entry point entity> by key | Access entry point entity using the key of th entity which is input to the enquiry. |
| Define set of <entry point entity> using input selection criteria | Define a set of <entry point entity>, each occurrence of which satisfies the input selection criteria. The operation syntax can be extended to define a particular order for the set, eg Order set in ascending <attribute>. |
| Read next <entity> in set | Access the next occurrence of <entity> whic forms part of the currently defined set. (NB: This operation must be preceded by a "Define set" operation.) |
| Read next occurrence of <detail entity> for the <master entity> | Access the next occurrence of <detail entity for the current occurrence of <master entity The operation syntax can be extended to specify a particular relationship, if there is more than one relationship between the twc entities. |
| Read <master entity> for <detail entity> | Access <master entity> for the current occurrence of <detail entity>. The operation syntax can be extended to specify a particul relationship, if there is more than one relationship between the two entities. |
| Read <entity> using <primary key attributes> | Access an occurrence of <entity> using its primary key. |
| Abort if <statement> | Abort the database process if a particular condition is satisfied. |
| Get <data items> | Obtain the values of particular data items which were input to the enquiry. |
| Output <data items> | Output data items to the function invoking the enquiry. |
| Fail if state indicator of <entity> outside <value range> | Report errors (or other relevant error processing) if the state indicator for an enti does not have a specific value. |
| Set state indicator of <entity> to <value> | Set the value of the state indicator of an entity to a specific value. |
| Write <entity> | Save all changes to the entity occurrence o the database. |

Operations are added to a data access or structure box in the logical sequence in which they are invoked. Typically, operations will be added in the following sequence:

1 : Operation to read an entity
2 : Operation to raise error if state indicator value is invalid
3 : Operation to create a new entity occurrence
4 : Operations taken from the ELH
5 : Operation to set the state indicator value
6 : Operations to output data to the function
7 : Operation to save/delete the entity occurrence

**NB: As part of tailoring SSADM, projects may decide not to document the individual operations on the ECDs. In such circumstances, projects may decide to merely annotate the data access/update boxes with an indication of the processing carried out by each box, eg I (Insert new occurrence), M (Modify existing occurrence), D (Delete existing occurrence), R (Read existing occurrence).**

The operations which will be taken from ELHs for inclusion on ECDs are as follows:



| Operation Type | Operation Description |
|---|---|
| Create <entity> | This operation is used for an effect which results in the creation of an entity occurrence. The following processing is implicit in this operation:<br>• setting values for the primary key using values which have been input as part of the event data;<br>• setting values for non-key attributes in the entity. Values will either be taken from the input event data or .set to default or "null" values. |
| Set <attribute> | This operation is used to change the value of an attribute using the value which has been input as part of the event data. |
| Set <attribute> using <expression> | This operation is used to set the value of an attribute to that derived from an expression/algorithm. *This operation type can be used as part of a create or modify effect.* |
| Delete <entity> | This operation is used for an effect which results in the deletion of an entity occurrence. |
| Tie to <master entity> | This operation is used to create a relationship between the entity occurrence being updated and an occurrence of a master entity. |
| Cut from <master entity> | This operation is used to remove a relationship between the entity occurrence being updated and an occurrence of a master entity. |
| Gain <detail entity> | This operation is used to create a relationship between the entity occurrence being updated and an occurrence of a detail entity. |
| Lose <detail entity> | This operation is used to remove a relationship between the entity occurrence being updated and an occurrence of a detail entity |
| Swap <master entity> | This operation is used to swap relationships from one occurrence of a master entity to another occurrence of the same master entity type. |
| Invoke <process> | This operation is used to invoke a process which is defined elsewhere, eg a common Elementary Process Description or a common enquiry. |

## Validate ECD

Each ECD must be validated to ensure that the data content of each of the entities accessed/updated can support the operations defined for the entities.

For each data access/update on the ECD ensure that all the data which is required by each operation is available. The required data must be available in one of the following ways:

- it must be present in the entity being accessed/updated;
- it must be present in one of the entities accessed **before** the entity currently being accessed/updated;
- it must be present as part of the input event data.
- it is generated by the process.

If an operation cannot be supported due to missing data, consider resolving this problem as follows:

- if data is needed from another entity, add an operation which invokes an enquiry process to obtain this data. If this enquiry process does not already exist, add it to the Entity Access Matrix and consider the impact of this new enquiry on the existing functions.
- if derived data is required, include the necessary derivation operations.

## ECD Worked Example

For the purposes of demonstrating how an ECD is produced, the following event will be used:

**Event Name: Decision of Court**

**Event Description:** This event is used to record the outcome of a court case (ie a verdict is reached).

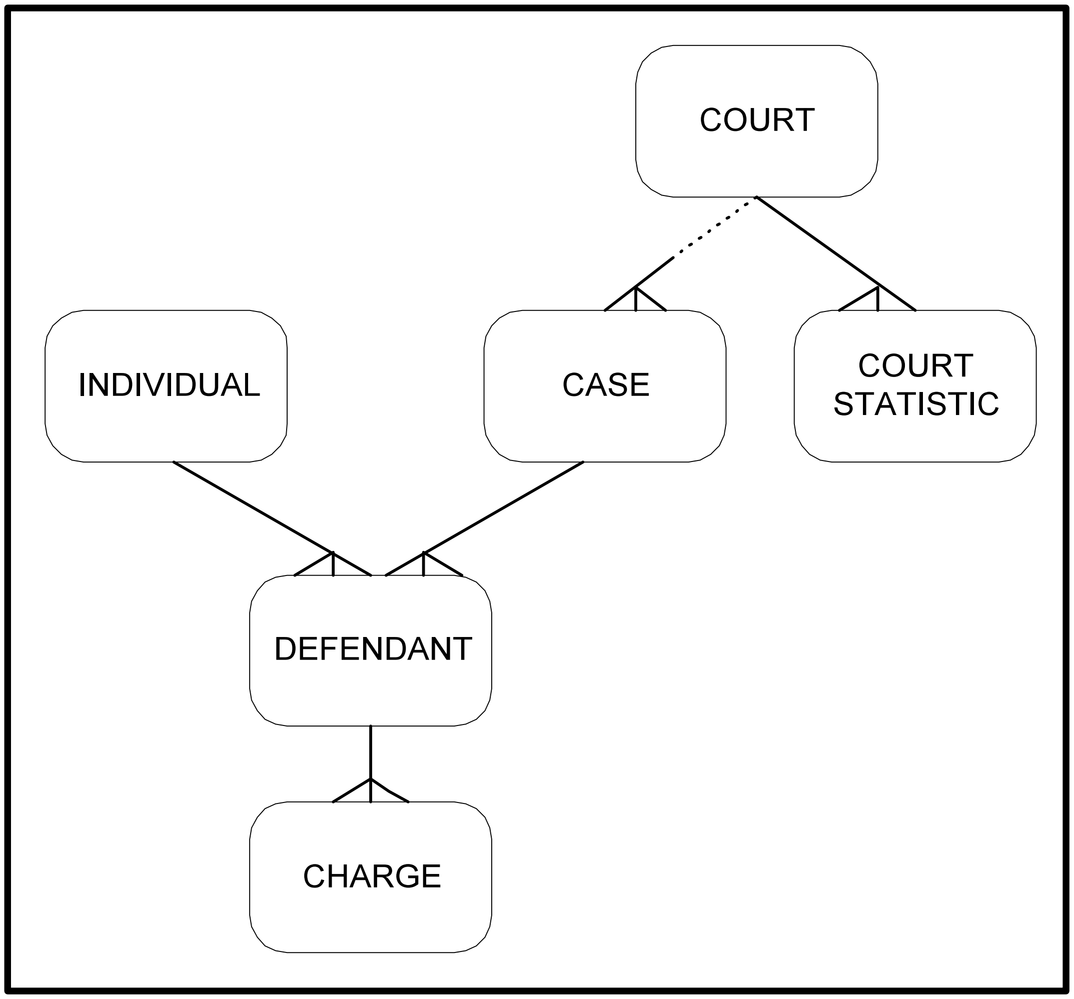
As a result of this event:

- the status of the case is set to 'Concluded'.
- the status of each defendant in the case is updated to either 'Convicted' or 'Cleared'.
- if a defendant is convicted, the status of each charge against the defendant is updated to either 'Convicted' or 'Cleared'.
- a count of the number of cases heard by the relevant court is incremented by 1.
- a count of the number of cases waiting to be heard by the court is decreased by 1.

**Entities Updated**

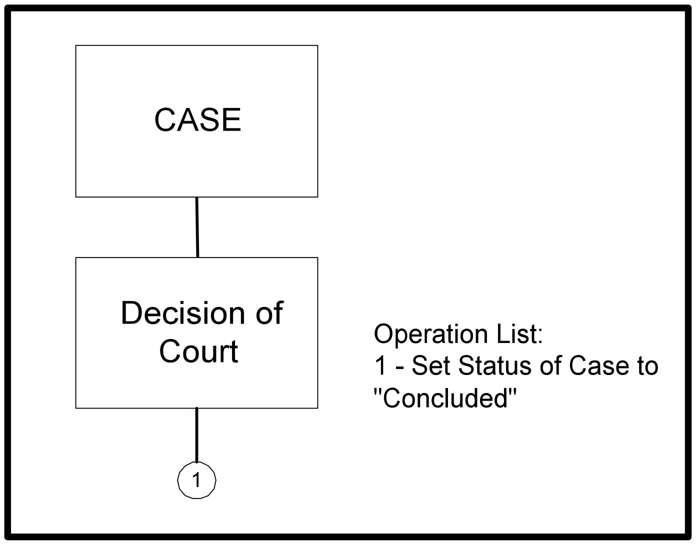
CASE - M
DEFENDANT (convicted) - M
DEFENDANT (cleared) - M
CHARGE (convicted) - M
CHARGE (cleared) - M
COURT STATISTIC [cases waiting] - M
COURT STATISTIC [cases heard] - M

The LDM extract which supports this event is as follows:

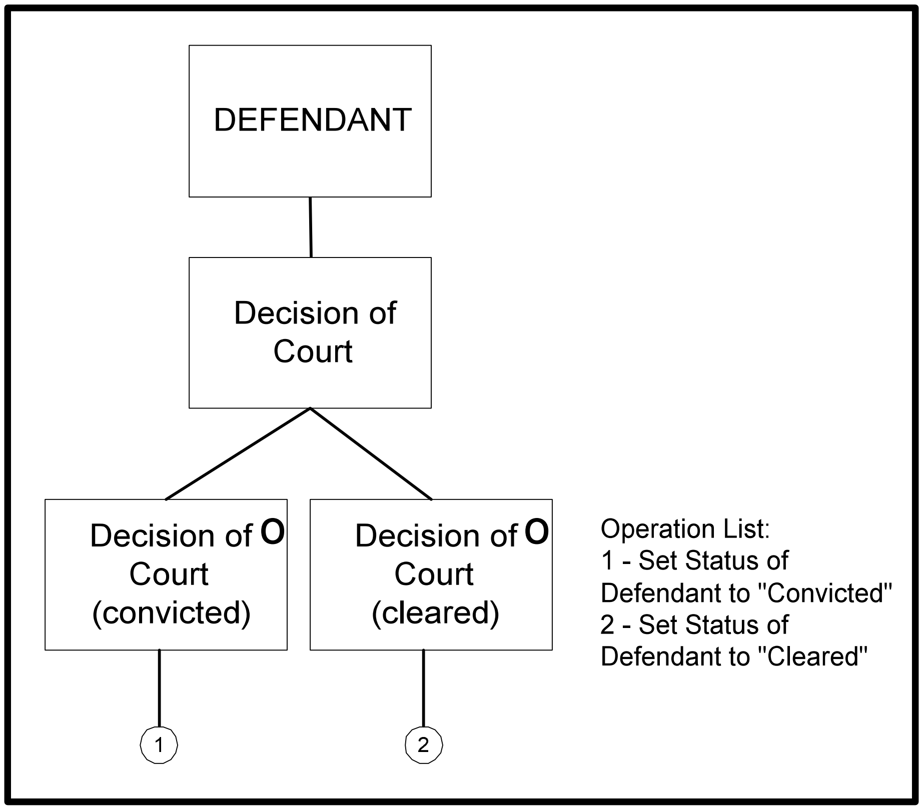

COURT
INDIVIDUAL
CASE
COURT STATISTIC
DEFENDANT
CHARGE

The four ELH extracts which are relevant for this worked example are as follows:

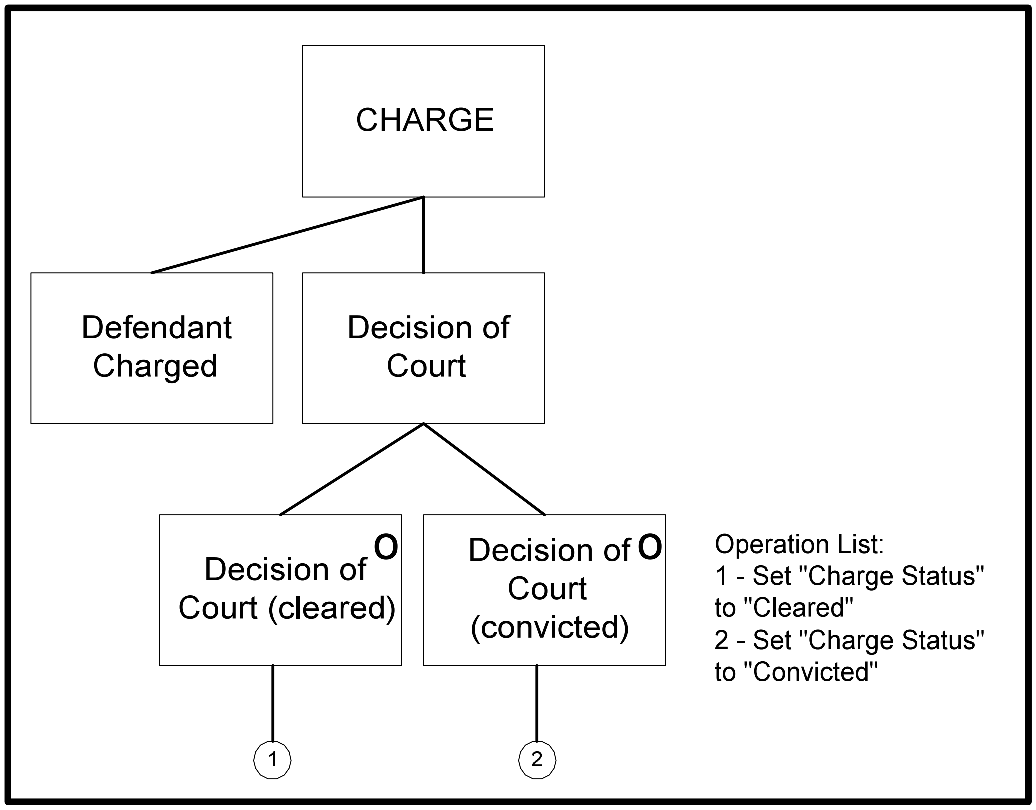
**ELH for CASE**



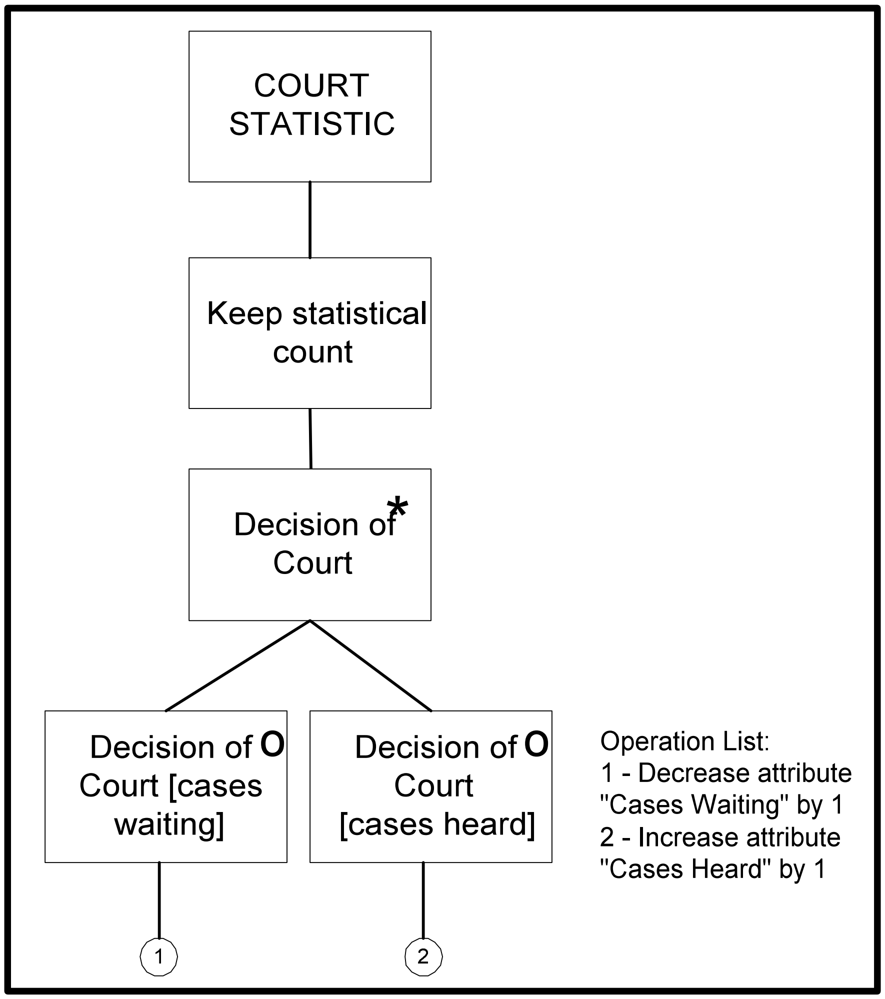
**ELH for DEFENDANT**


DEFENDANT
Decision of Court
Decision of Court (convicted) O
Decision of Court (cleared) O
1
2
Operation List:
1 - Set Status of Defendant to "Convicted"
2 - Set Status of Defendant to "Cleared"

**ELH for CHARGE**

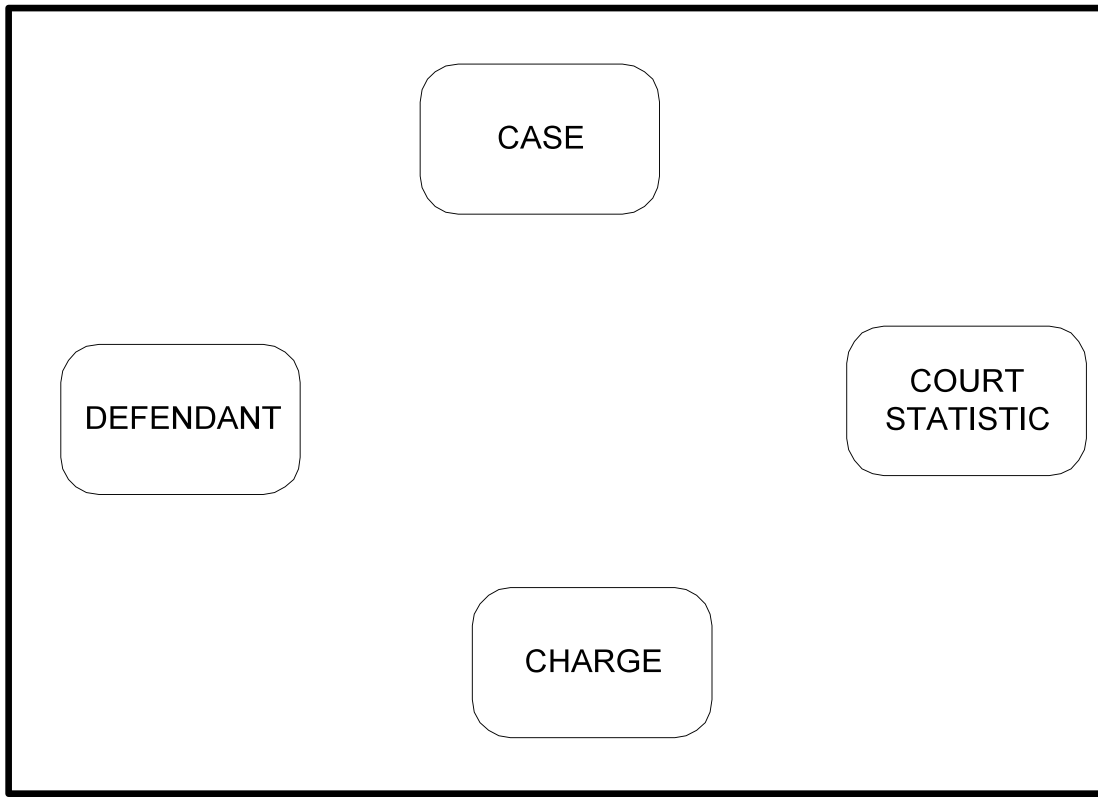

CHARGE
Defendant Charged
Decision of Court
Decision of Court (cleared) O
Decision of Court (convicted) O
1
2
Operation List:
1 - Set "Charge Status" to "Cleared"
2 - Set "Charge Status" to "Convicted"

**ELH for COURT STATISTIC**


COURT STATISTIC
Keep statistical count
Decision of Court *
Decision of Court [cases waiting] O
Decision of Court [cases heard] O
Operation List:
1 - Decrease attribute "Cases Waiting" by 1
2 - Increase attribute "Cases Heard" by 1
1
2

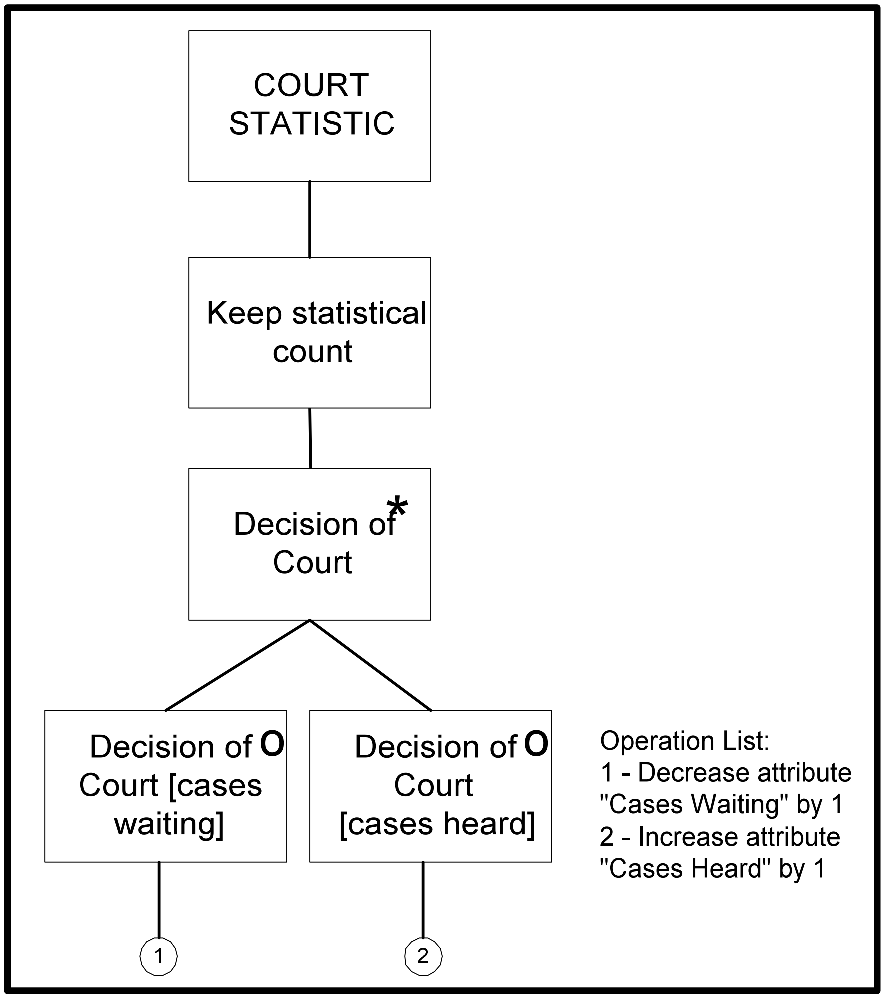
## ECD Worked Example: Draw a box for each entity affected by the event

For the Worked Example, the Event and Enquiry Catalogue shows that event *Decision of Court* updates the following entities CASE, DEFENDANT, CHARGE and COURT STATISTIC. ECD boxes are drawn for these entities as follows:



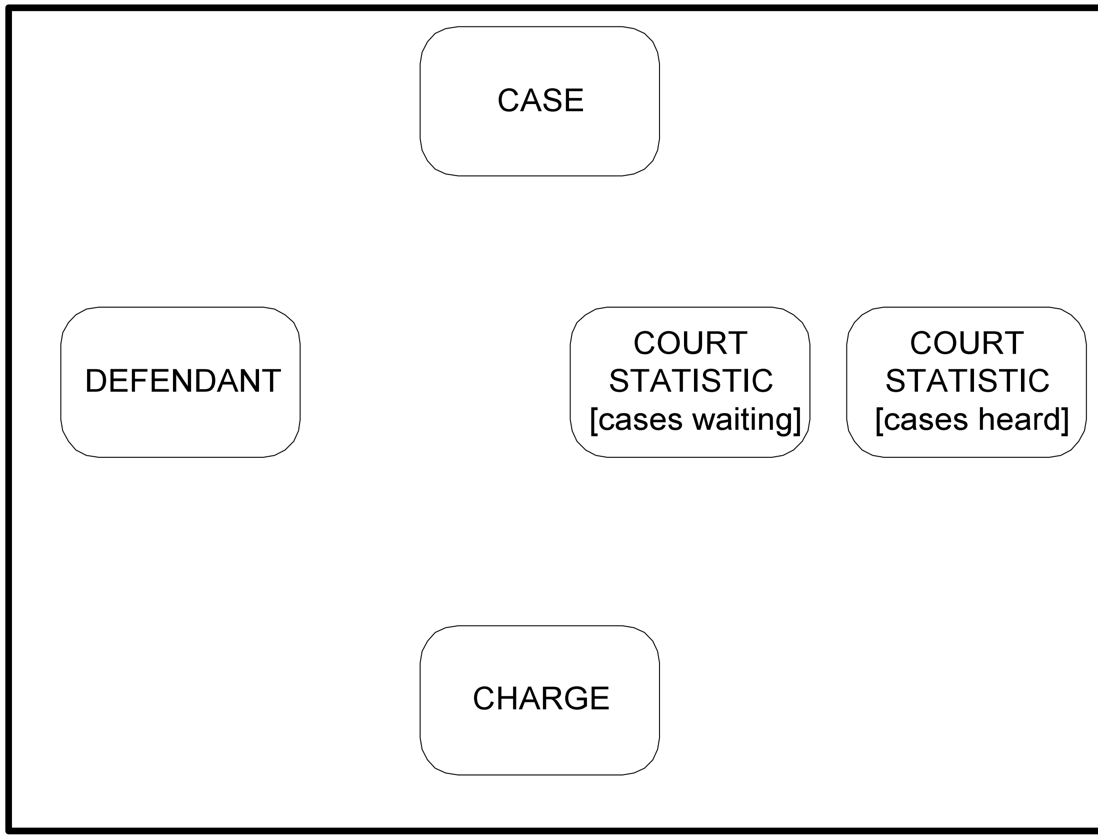
## ECD Worked Example: Add a box for each entity role

For the Worked Example, the ELH for entity COURT STATISTIC is as follows:


COURT STATISTIC
Keep statistical count
Decision of Court *
Decision of Court [cases waiting] O
Decision of Court [cases heard] O
1
2
Operation List:
1 - Decrease attribute "Cases Waiting" by 1
2 - Increase attribute "Cases Heard" by 1

The ELH for entity COURT STATISTIC shows that there are two roles for an instance of event *Decision of Court*. The occurrence of COURT STATISTIC which holds a count of the number of cases heard in the current year is incremented by 1. The occurrence of COURT STATISTIC which holds a count of the number of cases waiting to be heard is decreased by 1.

The two roles for entity COURT STATISTIC are included on the ECD as follows:

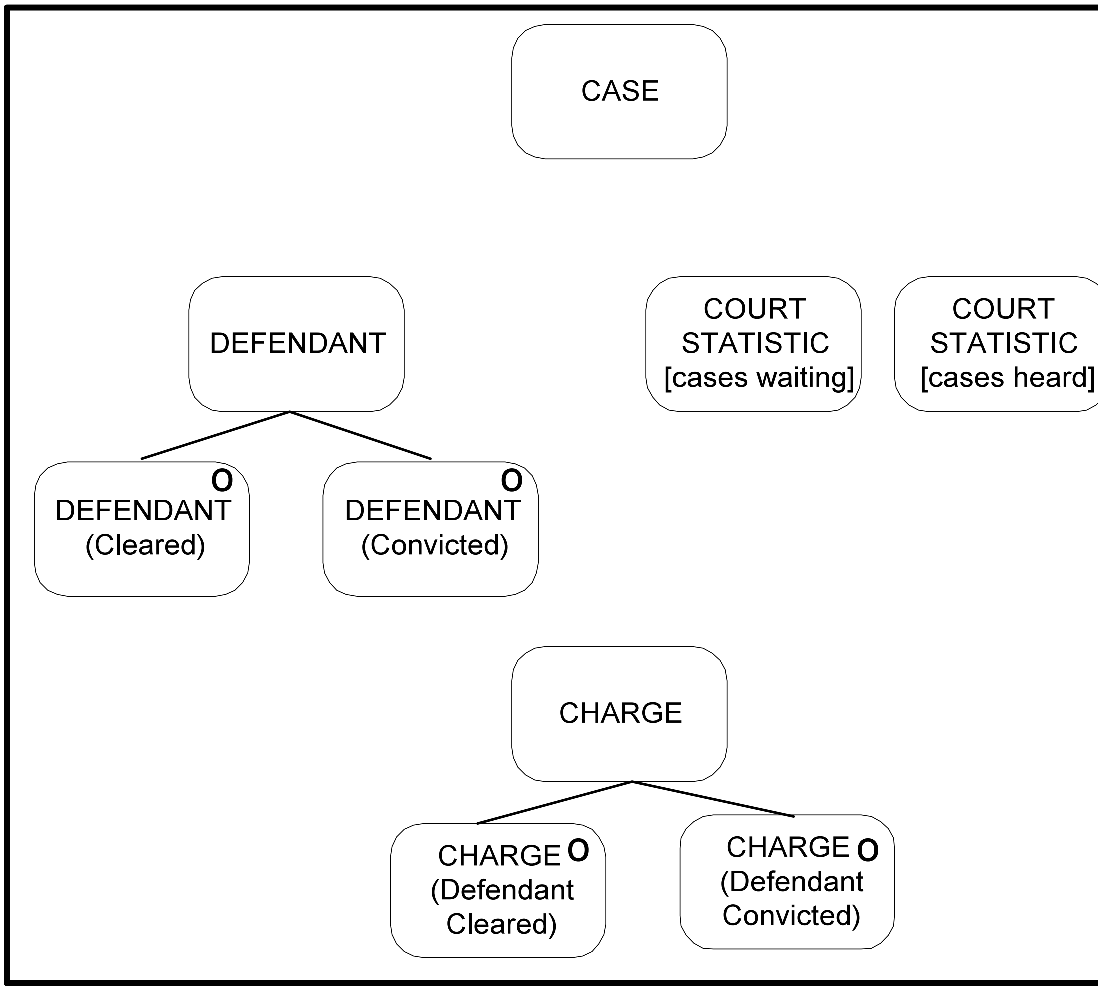

CASE
DEFENDANT
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
CHARGE

## ECD Worked Example: Add all selections

The ELHs for entities DEFENDANT and CHARGE show that the event *Decision of Court* has mutually exclusive effects on these entities.

- For entity DEFENDANT, the mutually exclusive effects depend on whether a defendant has been convicted or cleared.
- For entity CHARGE, the mutually exclusive effects depend on whether a defendant has been cleared or convicted.

The mutually exclusive effects are added to the ECD as follows:

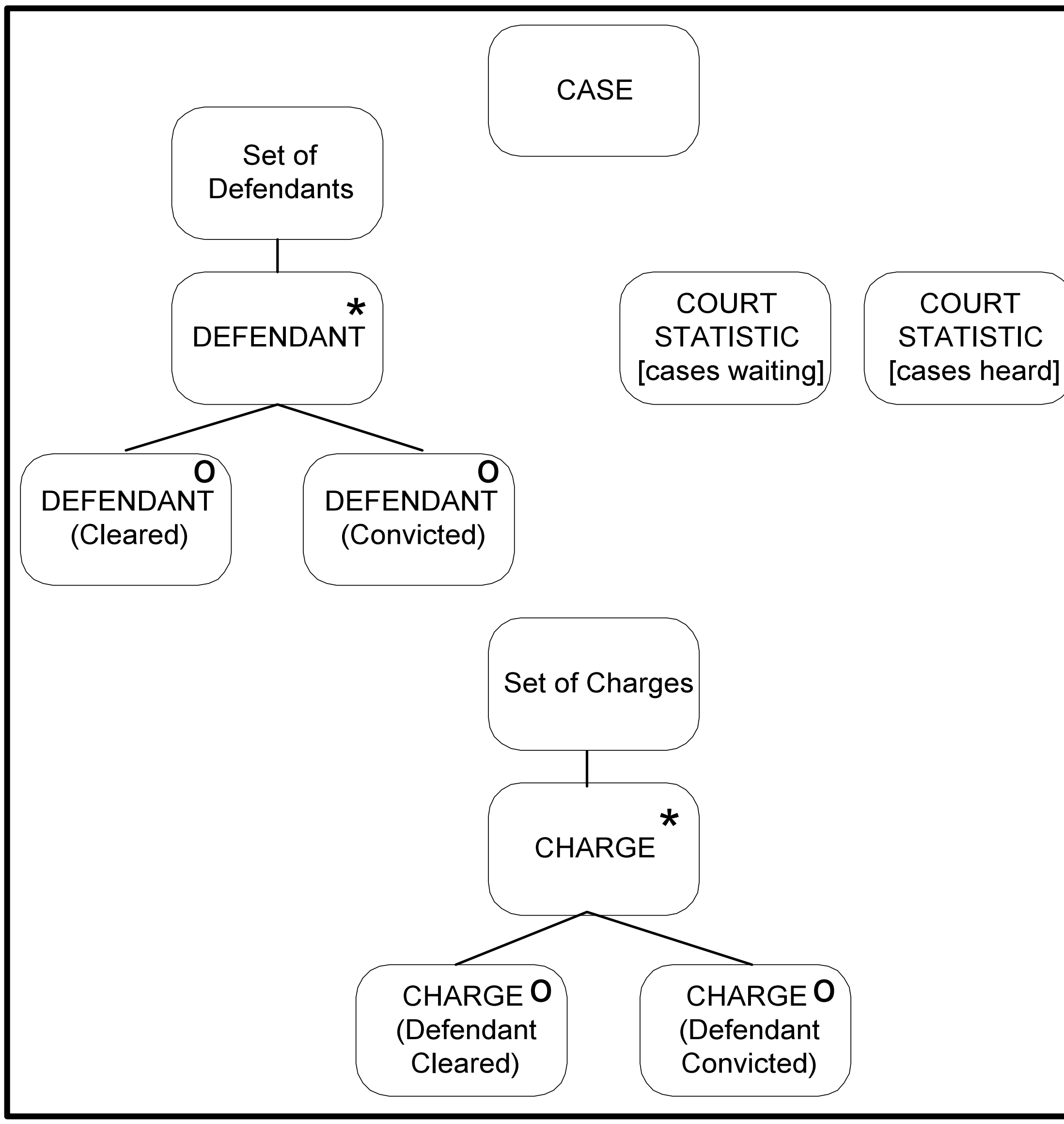

CASE
DEFENDANT
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
CHARGE
CHARGE O (Defendant Cleared)
CHARGE O (Defendant Convicted)

## ECD Worked Example: Add iteration structures

In the worked example, for each instance of event *Decision of Court*, the number of occurrences of each entity which is updated is potentially as follows:

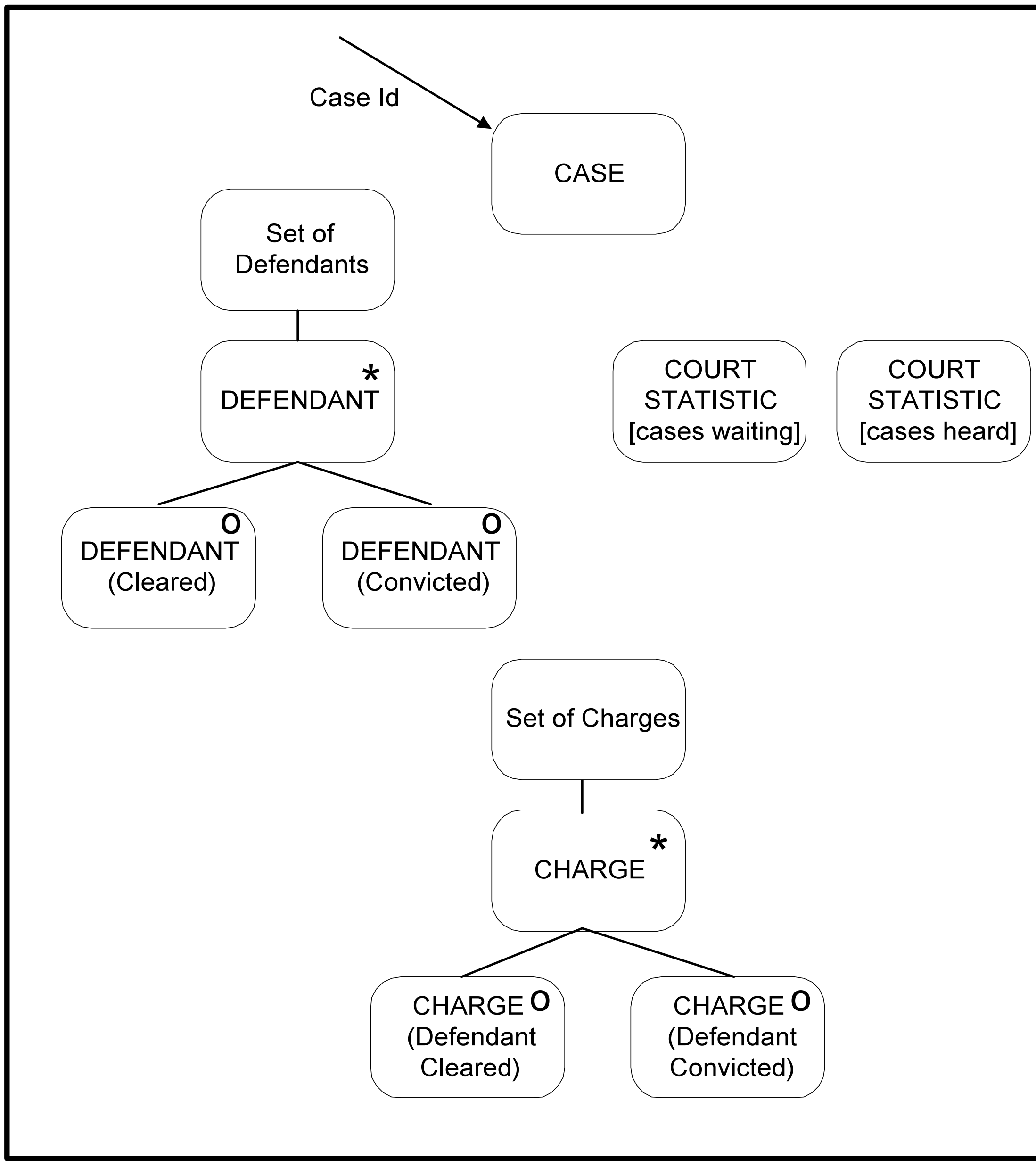
- Only 1 occurrence of entity CASE is updated.
- At most, multiple occurrences of entity DEFENDANT can be updated.
- At most, multiple occurrences of entity CHARGE can be updated.
- Only 1 occurrence of entity COURT STATISTIC in the role of 'Cases heard' is updated.
- Only 1 occurrence of entity COURT STATISTIC in the role of 'Cases waiting' is updated.

The ECD is updated to show possible iterations as follows:


CASE
Set of Defendants
DEFENDANT *
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
Set of Charges
CHARGE *
CHARGE O (Defendant Cleared)
CHARGE O (Defendant Convicted)

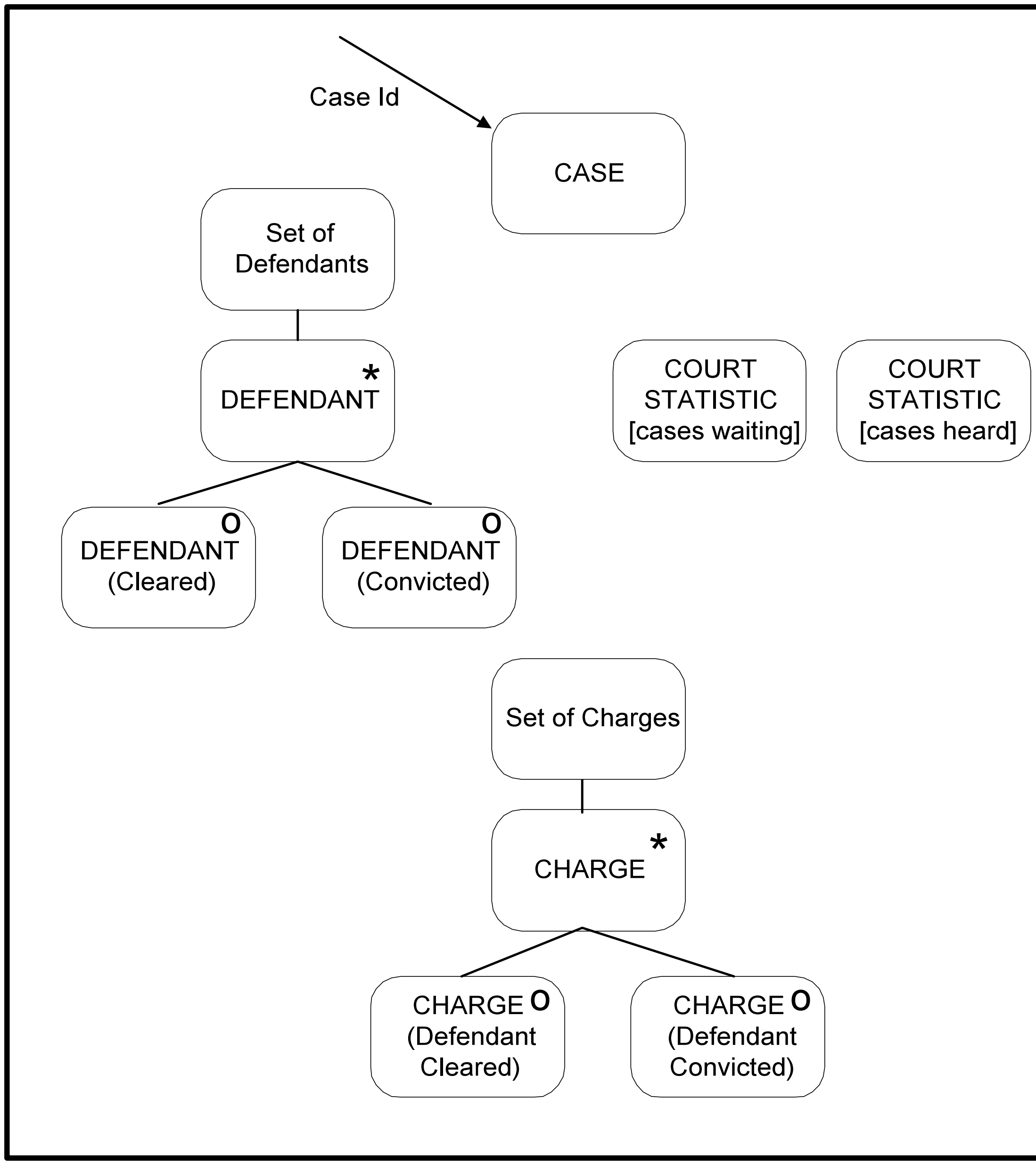
## ECD Worked Example: Identify entry point for event

The following diagram shows how the ECD has been enhanced to show that entity CASE is the entry point for event *Decision of Court*:

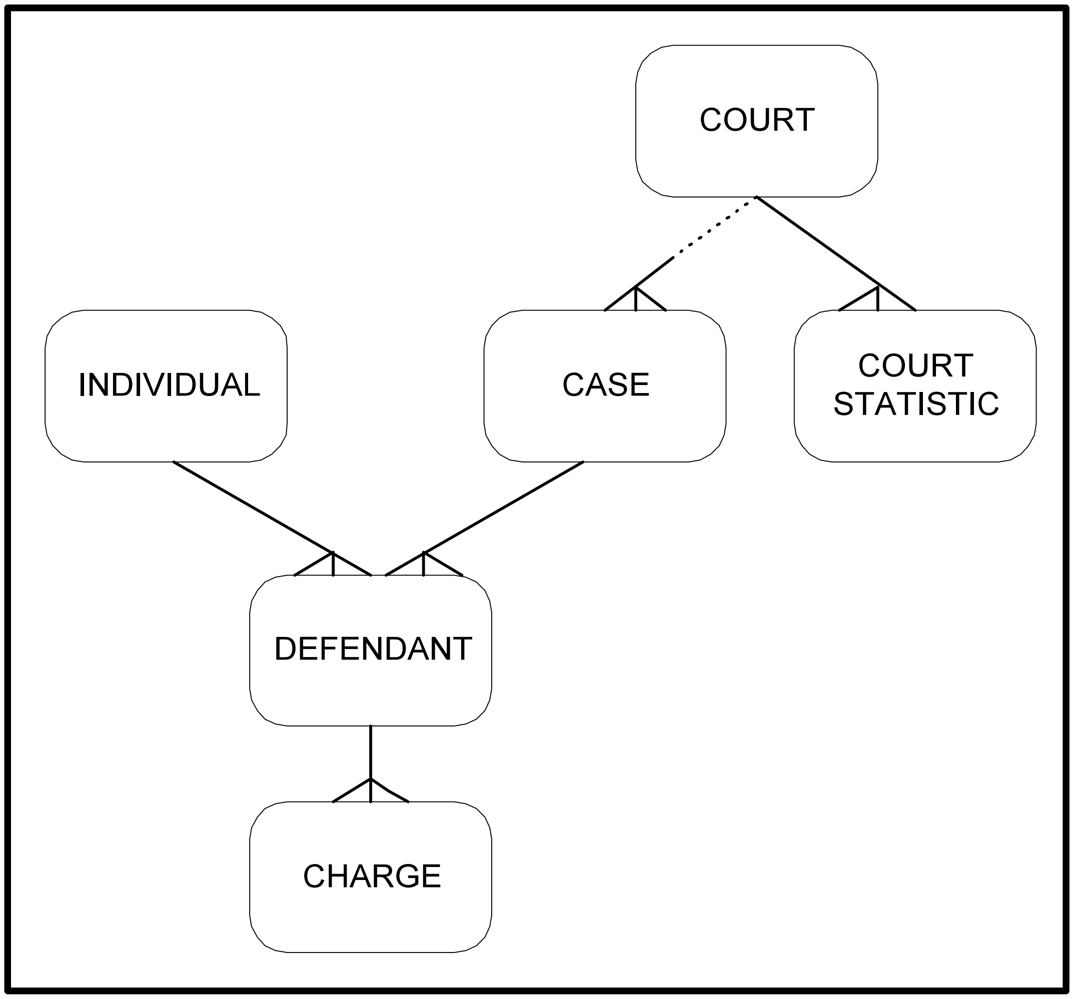

Case Id
CASE
Set of Defendants
DEFENDANT *
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
Set of Charges
CHARGE *
CHARGE (Defendant Cleared) O
CHARGE (Defendant Convicted) O

## ECD Worked Example: Define correspondences

The ECD so far is as follows:

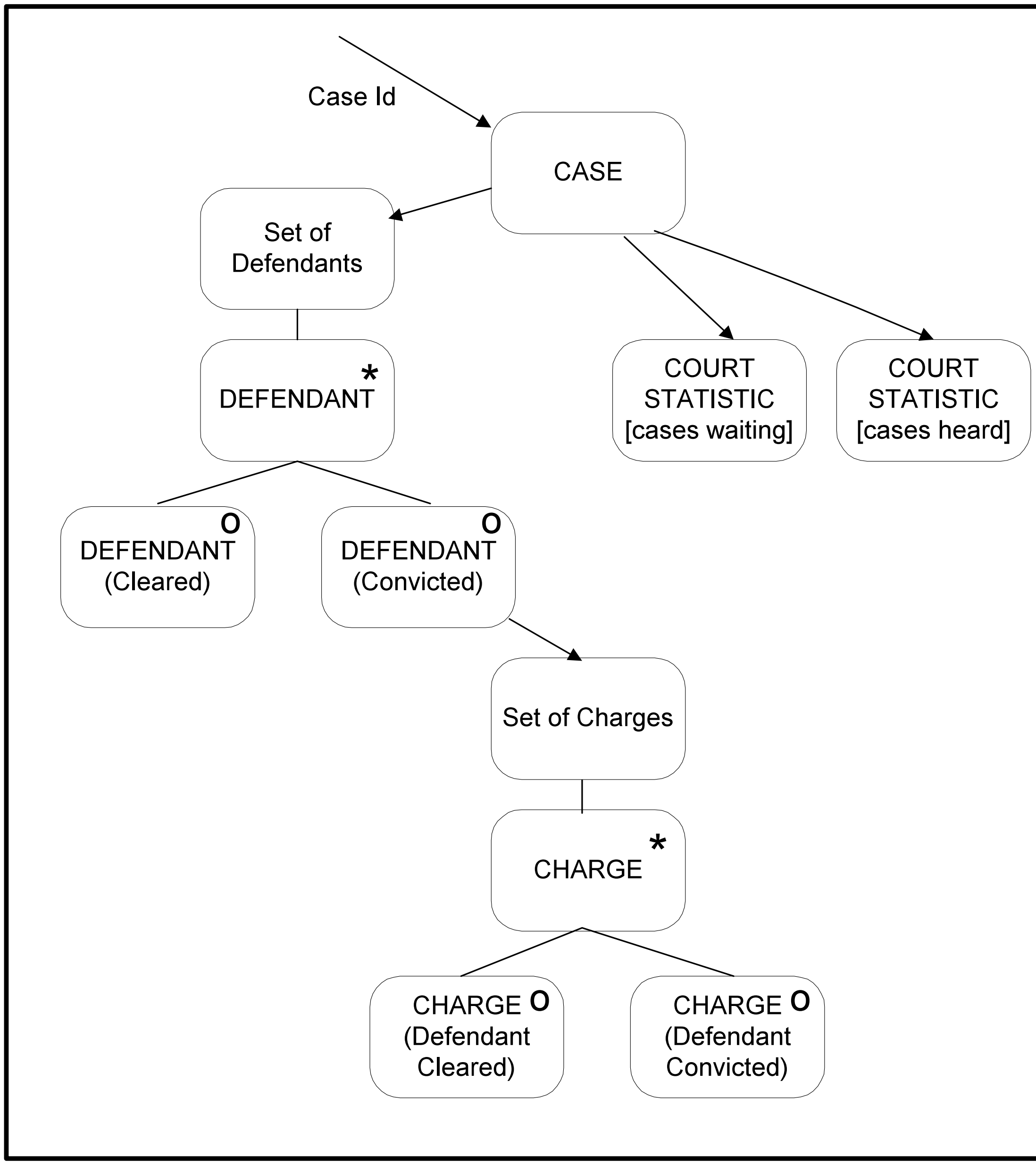

Case Id
CASE
Set of Defendants
DEFENDANT *
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
Set of Charges
CHARGE *
CHARGE O (Defendant Cleared)
CHARGE O (Defendant Convicted)

The LDM is as follows:



In navigating the LDM from the entry point entity, it appears at first glance that in order to get from entity CASE to entity COURT STATISTIC, entity COURT must be accessed, ie navigate from entity CASE to COURT and then to COURT STATISTIC. However, once entity CASE has been accessed, the event has values for the primary key of the two instances of entity COURT STATISTIC which are updated- attribute 'Court Name' is obtained from entity CASE and attribute 'Statistic Type' is generated by the event. Consequently, there is no need to access entity COURT in order to access entity COURT STATISTIC. The event process can go from entity CASE straight to entity COURT STATISTIC.

The ECD is then updated to show effect correspondences as follows:


Case Id
CASE
Set of Defendants
DEFENDANT *
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
DEFENDANT O (Cleared)
DEFENDANT O (Convicted)
Set of Charges
CHARGE *
CHARGE O (Defendant Cleared)
CHARGE O (Defendant Convicted)

The reason for each of the correspondences is as follows:

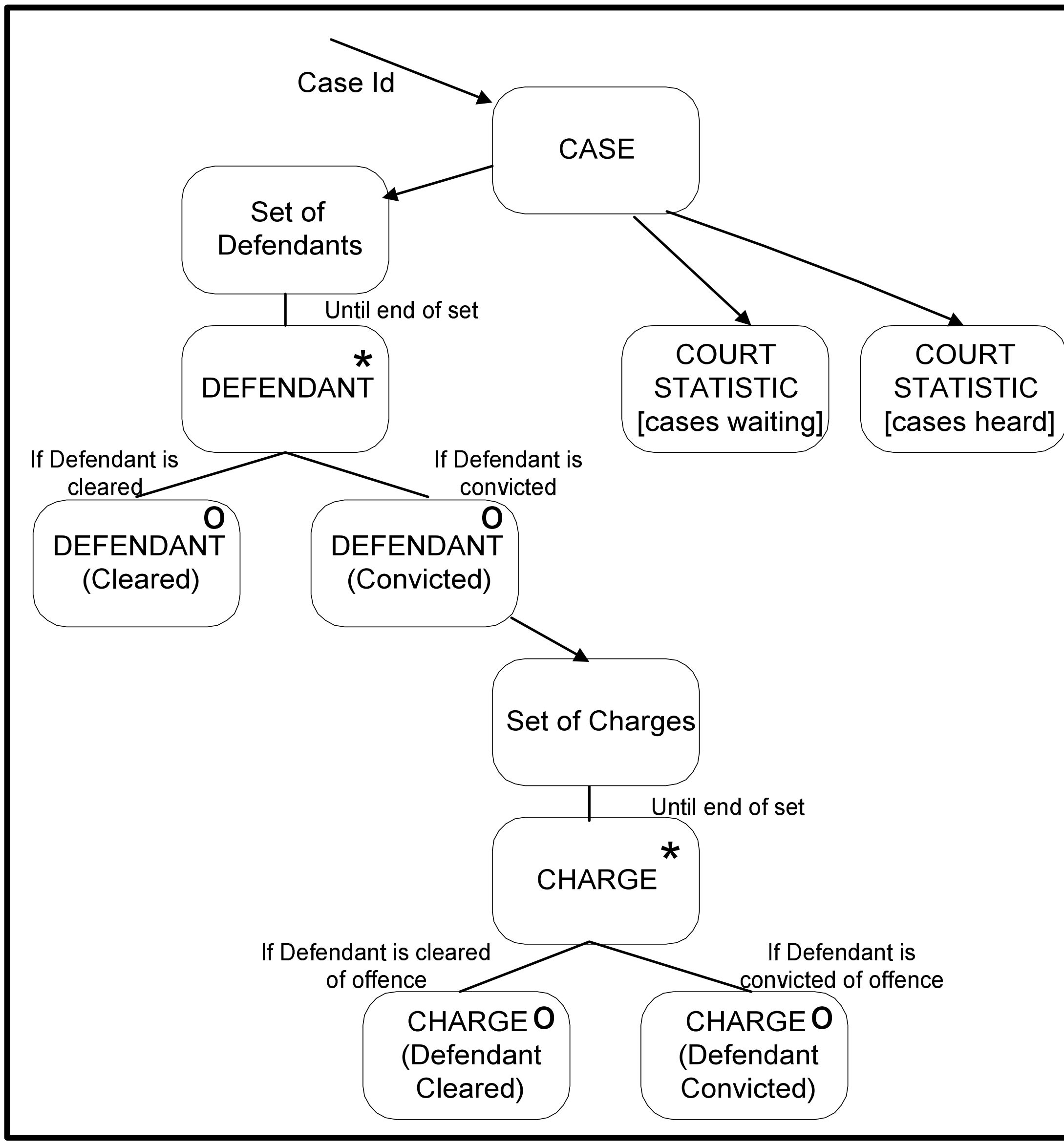
- CASE to COURT STATISTIC (Cases waiting) : Having accessed entity CASE and identified the court in which the case was heard (ie attribute Court Name), the occurrence of entity COURT STATISTIC which holds the number of cases waiting to be heard for that court can be accessed directly.
- CASE to COURT STATISTIC (Cases heard) : Having accessed entity CASE and identified the court in which the case was heard (ie attribute Court

Name), the occurrence of entity COURT STATISTIC which holds the number of cases heard by that court can be accessed directly.

- CASE to Set of Defendants : The defendants for a case are accessed and the status updated to either 'Cleared' or 'Convicted'.
- DEFENDANT (Convicted) and Set of Charges : For each convicted defendant in a case, each of the associated charges must be updated. Correspondence is therefore shown between DEFENDANT (Convicted) and Set of Charges.

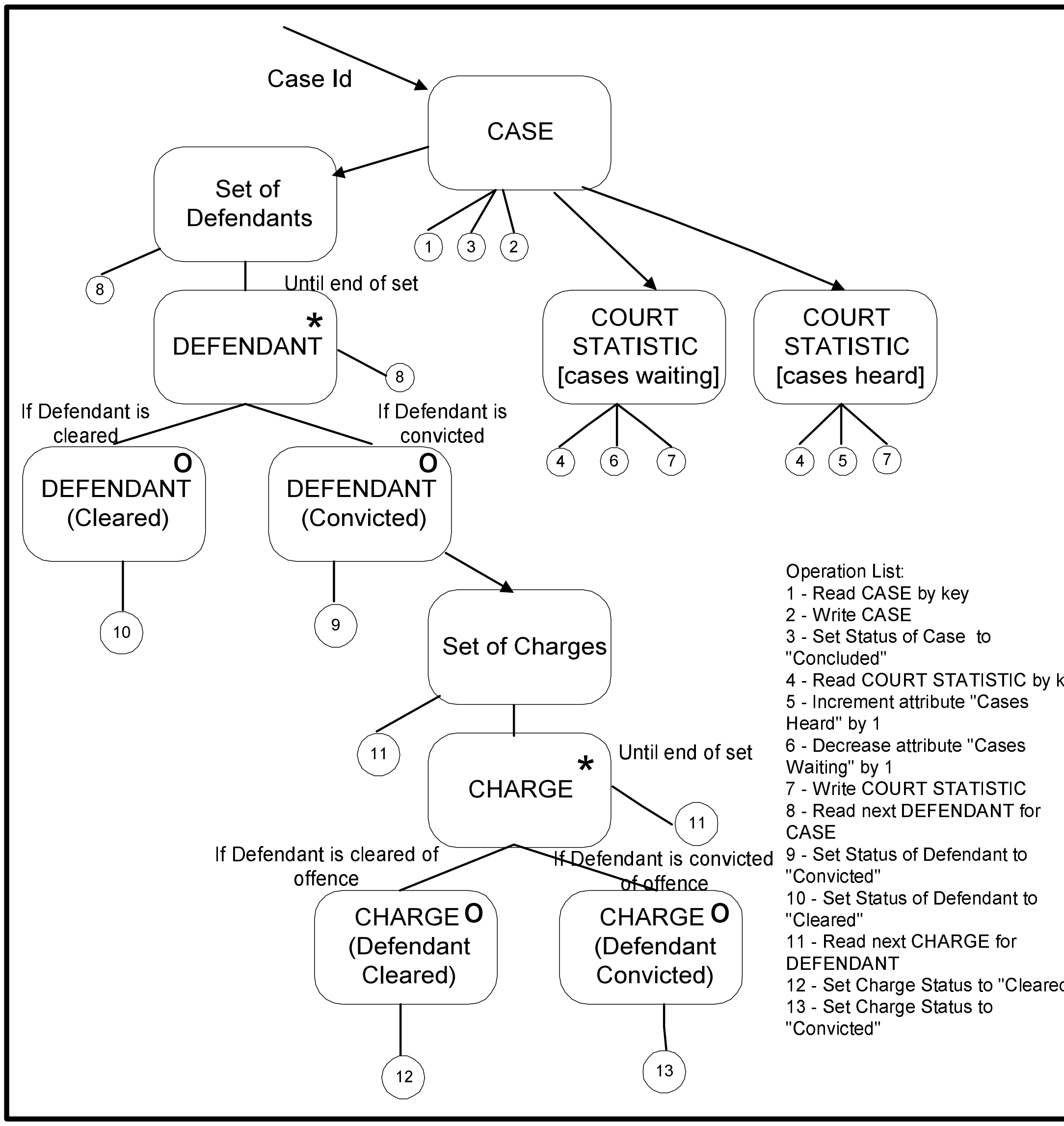
## ECD Worked Example: Add conditions to selections and iterations

The conditions governing the iterations and selections are added to the ECD as follows:


Case Id
CASE
Set of Defendants
Until end of set
DEFENDANT *
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
If Defendant is cleared
If Defendant is convicted
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
Set of Charges
Until end of set
CHARGE *
If Defendant is cleared of offence
If Defendant is convicted of offence
CHARGE O (Defendant Cleared)
CHARGE O (Defendant Convicted)

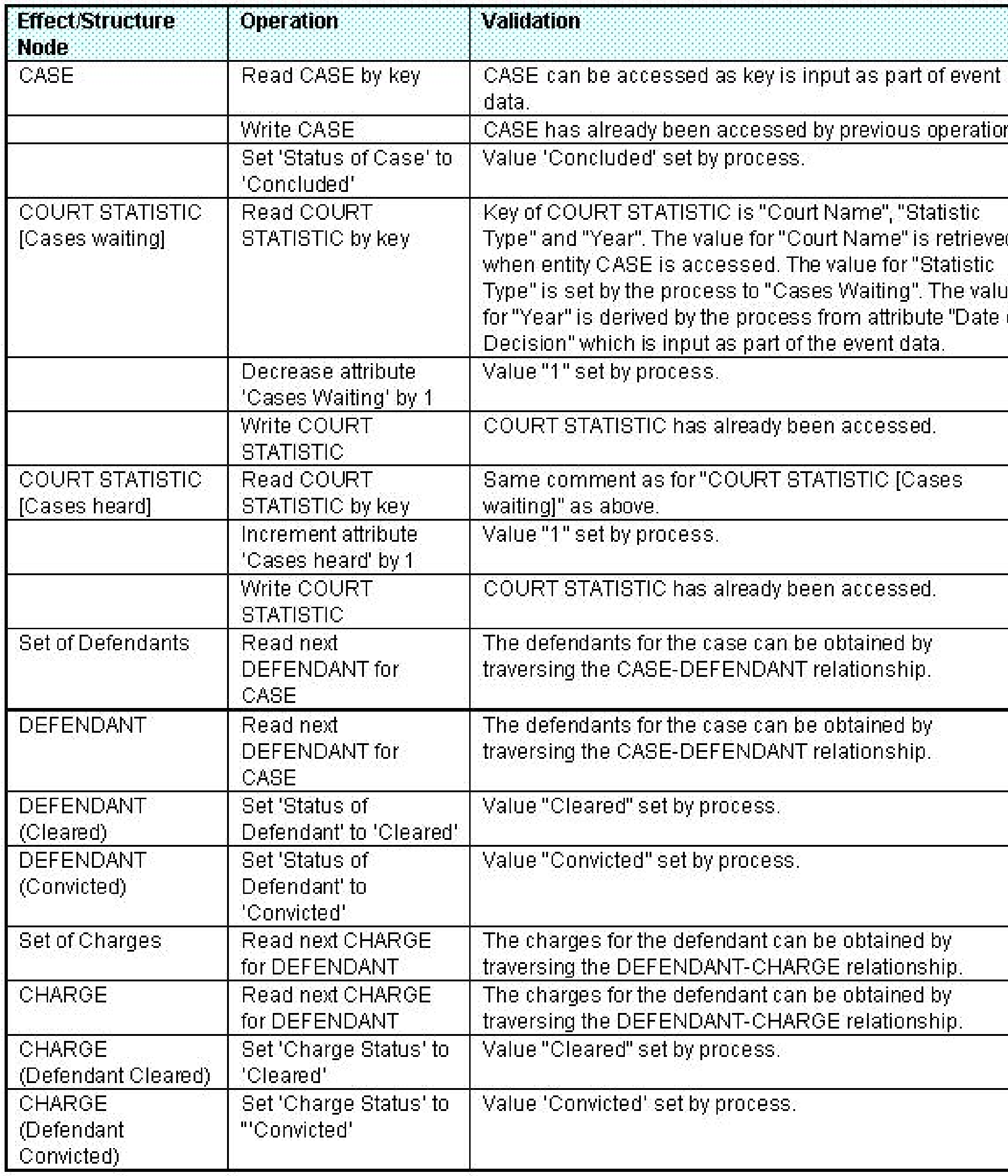
## ECD Worked Example: Validate ECD

The ECD for event *Decision of Court* is as follows:


Case Id
CASE
Set of Defendants
1
3
2
8
Until end of set
DEFENDANT *
8
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
If Defendant is cleared
If Defendant is convicted
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
4
6
7
4
5
7
10
9
Set of Charges
11
Until end of set
CHARGE *
11
If Defendant is cleared of offence
If Defendant is convicted of offence
CHARGE O (Defendant Cleared)
CHARGE O (Defendant Convicted)
12
13

Operation List:
1 - Read CASE by key
2 - Write CASE
3 - Set Status of Case to "Concluded"
4 - Read COURT STATISTIC by k
5 - Increment attribute "Cases Heard" by 1
6 - Decrease attribute "Cases Waiting" by 1
7 - Write COURT STATISTIC
8 - Read next DEFENDANT for CASE
9 - Set Status of Defendant to "Convicted"
10 - Set Status of Defendant to "Cleared"
11 - Read next CHARGE for DEFENDANT
12 - Set Charge Status to "Clearec
13 - Set Charge Status to "Convicted"

The ECD is validated as follows:



| Effect/Structure Node | Operation | Validation |
|---|---|---|
| CASE | Read CASE by key | CASE can be accessed as key is input as part of event data. |
| | Write CASE | CASE has already been accessed by previous operatior |
| | Set 'Status of Case' to 'Concluded' | Value 'Concluded' set by process. |
| COURT STATISTIC [Cases waiting] | Read COURT STATISTIC by key | Key of COURT STATISTIC is "Court Name", "Statistic Type" and "Year". The value for "Court Name" is retrievec when entity CASE is accessed. The value for "Statistic Type" is set by the process to "Cases Waiting". The valu for "Year" is derived by the process from attribute "Date Decision" which is input as part of the event data. |
| | Decrease attribute 'Cases Waiting' by 1 | Value "1" set by process. |
| | Write COURT STATISTIC | COURT STATISTIC has already been accessed. |
| COURT STATISTIC [Cases heard] | Read COURT STATISTIC by key | Same comment as for "COURT STATISTIC [Cases waiting]" as above. |
| | Increment attribute 'Cases heard' by 1 | Value "1" set by process. |
| | Write COURT STATISTIC | COURT STATISTIC has already been accessed. |
| Set of Defendants | Read next DEFENDANT for CASE | The defendants for the case can be obtained by traversing the CASE-DEFENDANT relationship. |
| DEFENDANT | Read next DEFENDANT for CASE | The defendants for the case can be obtained by traversing the CASE-DEFENDANT relationship. |
| DEFENDANT (Cleared) | Set 'Status of Defendant' to 'Cleared' | Value "Cleared" set by process. |
| DEFENDANT (Convicted) | Set 'Status of Defendant' to 'Convicted' | Value "Convicted" set by process. |
| Set of Charges | Read next CHARGE for DEFENDANT | The charges for the defendant can be obtained by traversing the DEFENDANT-CHARGE relationship. |
| CHARGE | Read next CHARGE for DEFENDANT | The charges for the defendant can be obtained by traversing the DEFENDANT-CHARGE relationship. |
| CHARGE (Defendant Cleared) | Set 'Charge Status' to 'Cleared' | Value "Cleared" set by process. |
| CHARGE (Defendant Convicted) | Set 'Charge Status' to "'Convicted' | Value 'Convicted' set by process. |

**NB: The above table has been produced purely for the purposes of this worked example. Such formal validation of ECD operations is not required by SSADM.**

## Enquiry Process Models

The steps involved in transforming an Enquiry Access Path into an Enquiry Process Model are as follows:

- **Group accesses on EAP**
- **Convert grouped accesses into Jackson-like notation**
- **Allocate operations and conditions to Enquiry Process Model**
- **Validate the Enquiry Process Model**

A worked example demonstrating each of the above steps is provided.

## Group accesses on EAP

The accesses and structure boxes which are joined by one-to-one navigation arrows on the EAP are grouped together. If an access or structure box is not joined by a one-to-one navigation arrow with any part of the EAP, the access/structure box constitutes a grouping by itself.

## Convert grouped accesses into Jackson-like notation

The grouped accesses on the EAP are converted into a Jackson-like structure. Each group of accesses becomes either a structure box or bottom-leaf box on the Jackson-like structure. Additional structure boxes may need to be added to adhere to the Jackson structuring rules.

Each group of accesses is given a name which reflects the overall processing represented by the grouping.

## Allocate operations and conditions to Enquiry Process Model

The operations and conditions on the EAP should be transferred to the relevant EPM nodes, ie those nodes which contain the EAP data accesses to which the operations and conditions relate.

## Validate the Enquiry Process Model

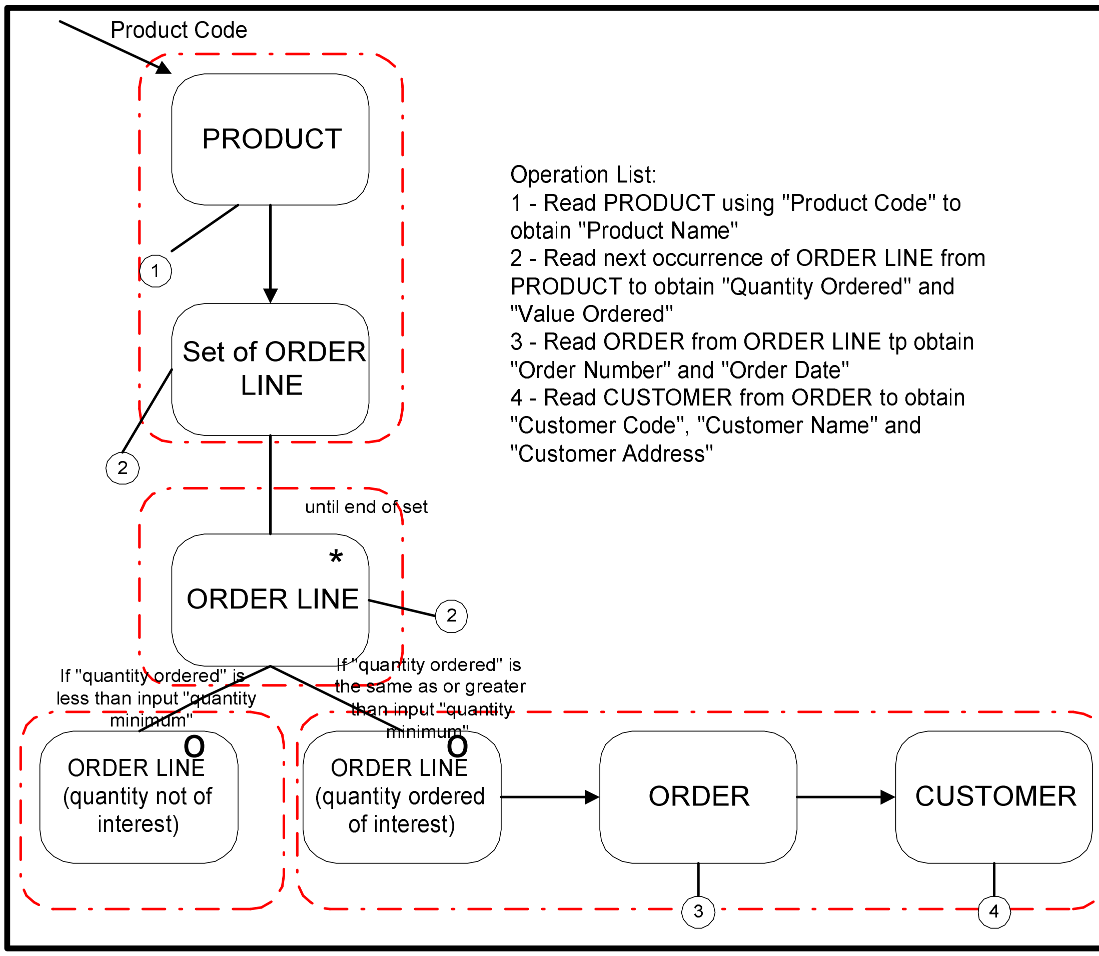
The Enquiry Process Model must be checked to ensure that the processing and sequence of processing defined makes sense. The resultant Process Model should be walked through with the objective of identifying any incorrect sequencing, iteration or selection of operations.

## EPM Worked example

The EAP for enquiry “Produce Product History List” which was produced earlier will now be developed into an Enquiry Process Model.

## EPM Worked example: Group accesses on EAP

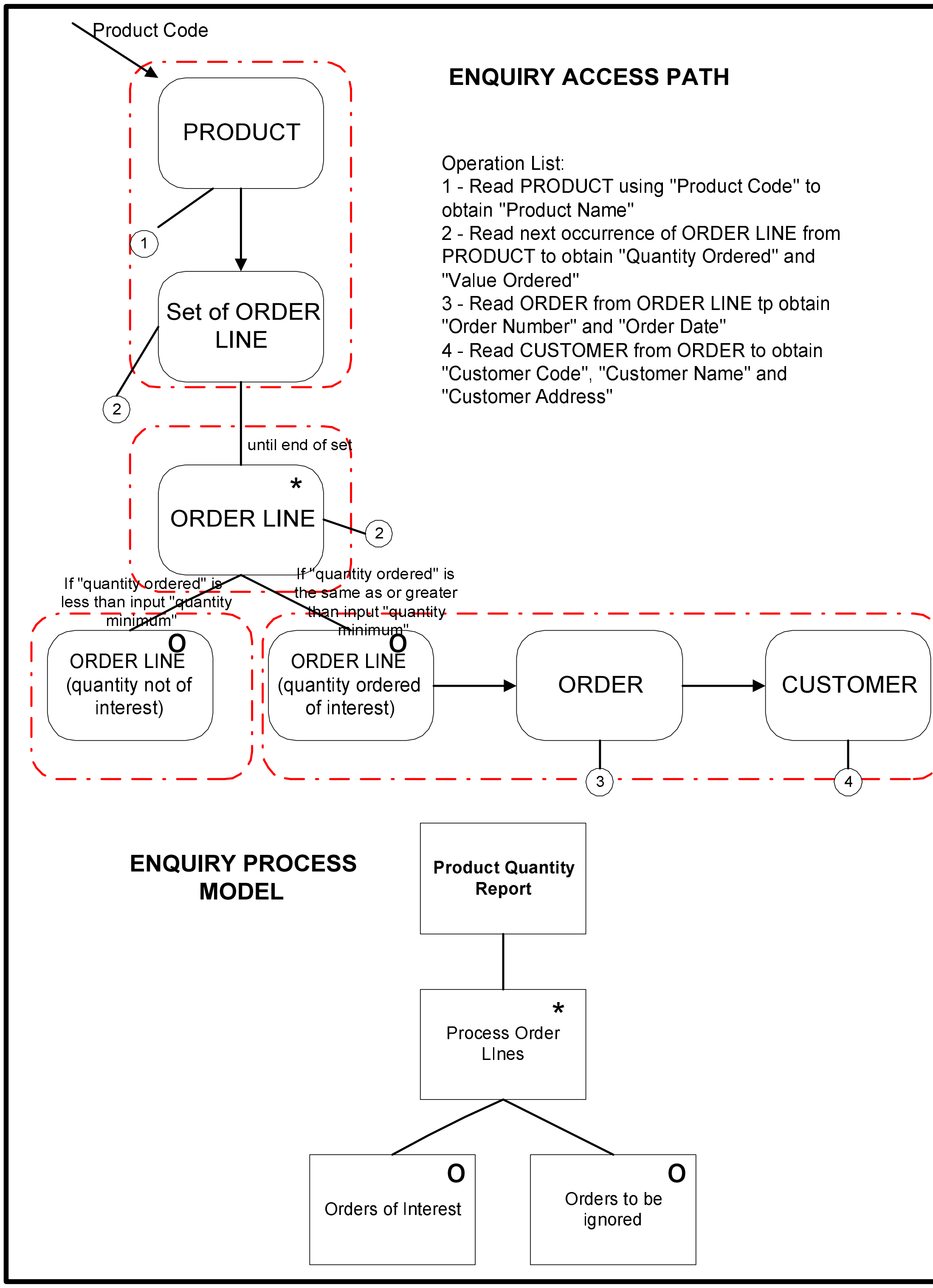
The grouping of accesses/structures on the EAP “Produce Product History List” is shown below.


Product Code
PRODUCT
1
Set of ORDER LINE
2
until end of set
*
ORDER LINE
2
If "quantity ordered" is less than input "quantity minimum"
If "quantity ordered" is the same as or greater than input "quantity minimum"
O
ORDER LINE (quantity not of interest)
O
ORDER LINE (quantity ordered of interest)
ORDER
CUSTOMER
3
4

Operation List:
1 - Read PRODUCT using "Product Code" to obtain "Product Name"
2 - Read next occurrence of ORDER LINE from PRODUCT to obtain "Quantity Ordered" and "Value Ordered"
3 - Read ORDER from ORDER LINE tp obtain "Order Number" and "Order Date"
4 - Read CUSTOMER from ORDER to obtain "Customer Code", "Customer Name" and "Customer Address"

## EPM Worked example: Convert grouped accesses into Jackson-like notation

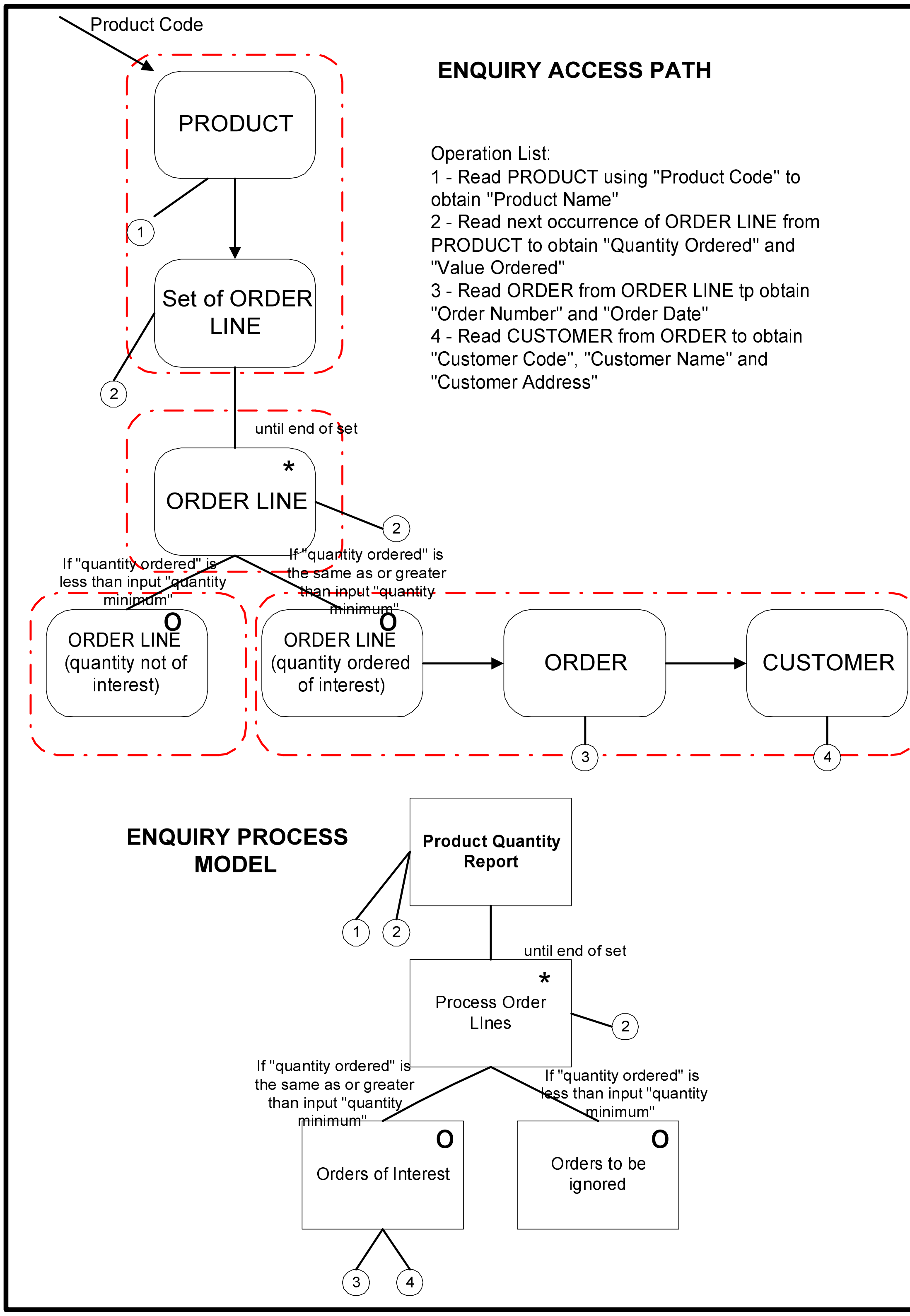
The grouped accesses which need to be converted to a Jackson-like structure and the resultant structure is shown below.


Product Code
PRODUCT
1
Set of ORDER LINE
2
until end of set
ORDER LINE *
2
If "quantity ordered" is less than input "quantity minimum"
If "quantity ordered" is the same as or greater than input "quantity minimum"
ORDER LINE (quantity not of interest) O
ORDER LINE (quantity ordered of interest) O
ORDER
CUSTOMER
3
4
ENQUIRY ACCESS PATH
ENQUIRY PROCESS MODEL
Product Quantity Report
Process Order LInes *
Orders of Interest O
Orders to be ignored O

Operation List:
1 - Read PRODUCT using "Product Code" to obtain "Product Name"
2 - Read next occurrence of ORDER LINE from PRODUCT to obtain "Quantity Ordered" and "Value Ordered"
3 - Read ORDER from ORDER LINE tp obtain "Order Number" and "Order Date"
4 - Read CUSTOMER from ORDER to obtain "Customer Code", "Customer Name" and "Customer Address"

## EPM Worked example: Allocate operations and conditions to Enquiry Process Model

The allocation of operations and conditions to the EPM is shown in below:


Product Code
ENQUIRY ACCESS PATH
PRODUCT
Set of ORDER LINE
until end of set
*
ORDER LINE
If "quantity ordered" is less than input "quantity minimum"
If "quantity ordered" is the same as or greater than input "quantity minimum"
O
ORDER LINE (quantity not of interest)
O
ORDER LINE (quantity ordered of interest)
ORDER
CUSTOMER
1
2
2
3
4
Operation List:
1 - Read PRODUCT using "Product Code" to obtain "Product Name"
2 - Read next occurrence of ORDER LINE from PRODUCT to obtain "Quantity Ordered" and "Value Ordered"
3 - Read ORDER from ORDER LINE tp obtain "Order Number" and "Order Date"
4 - Read CUSTOMER from ORDER to obtain "Customer Code", "Customer Name" and "Customer Address"
ENQUIRY PROCESS MODEL
Product Quantity Report
1
2
until end of set
*
Process Order LInes
2
If "quantity ordered" is the same as or greater than input "quantity minimum"
If "quantity ordered" is less than input "quantity minimum"
O
Orders of Interest
O
Orders to be ignored
3
4

## Update Process Models

The steps involved in transforming an Effect Correspondence Diagram into an Update Process Model are as follows:

- **Group effects on ECD**
- **Convert grouped effects into Jackson-like notation**
- **Allocate operations and conditions to Update Process Model**
- **Validate the Update Process Model**

A worked example demonstrating each of the above steps is provided.

## Group effects on ECD

The effects and structure boxes which are joined by one-to-one correspondence arrows on the ECD are grouped together. If an effect or structure box is not joined by a one-to-one correspondence arrow with any other part of the ECD, the effect/structure box constitutes a grouping by itself.

## Convert grouped effects into Jackson-like notation

The grouped effects on the ECD are converted into a Jackson-like structure. Each group of effects becomes either a structure box or bottom-leaf box on the Jackson-like structure. Additional structure boxes may need to be added to adhere to the Jackson structuring rules.

Each group of effects is given a name which reflects the overall processing represented by the grouping.

## Allocate operations and conditions to Update Process Model

The operations and conditions on the ECD should be transferred to the relevant UPM nodes, ie those nodes which contain the ECD effects to which the operations and conditions relate.

## Validate the Update Process Model

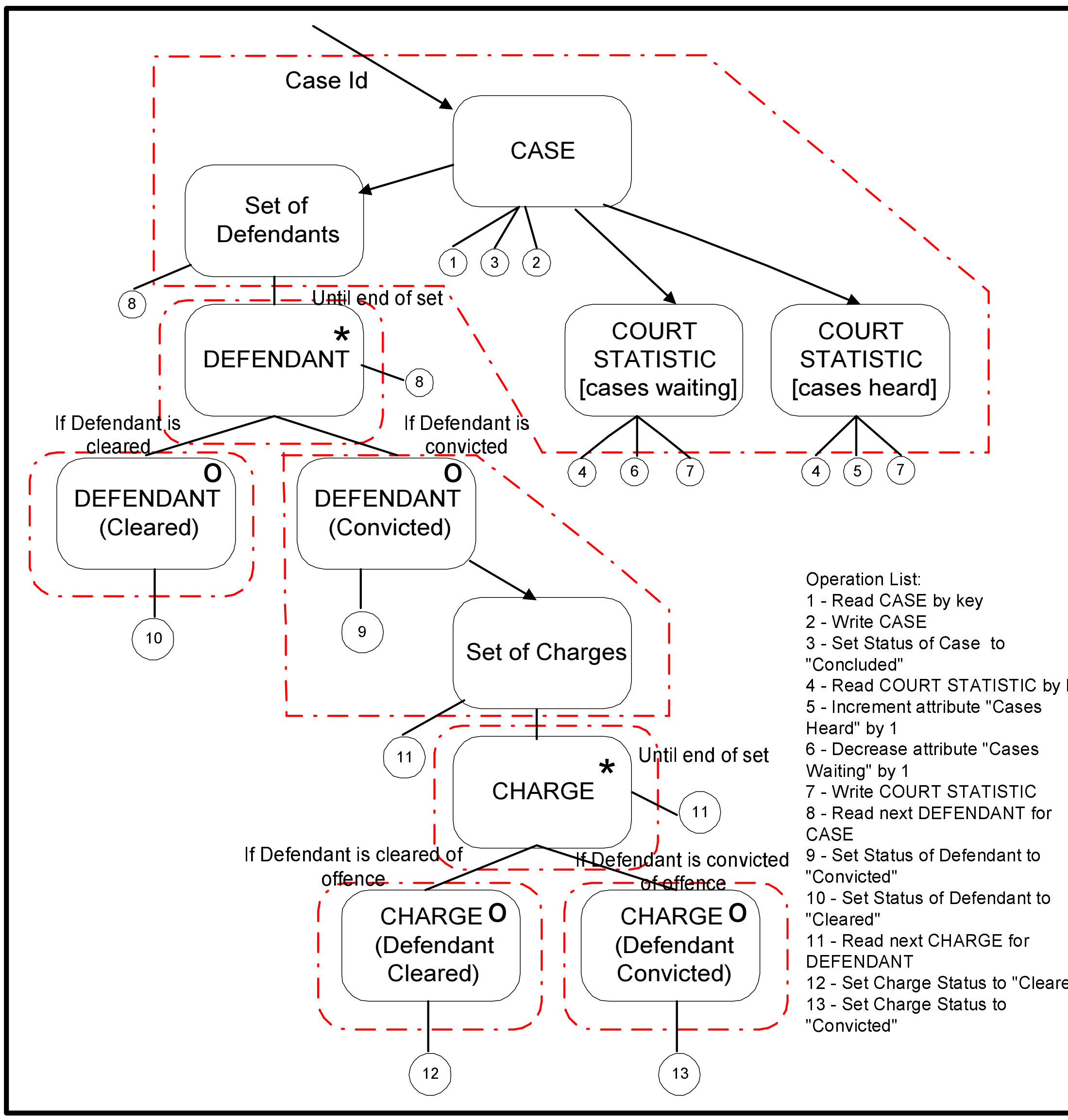
The Update Process Model must be checked to ensure that the processing and sequence of processing defined makes sense. The resultant Process Model should be walked through with the objective of identifying any incorrect sequencing, iteration or selection of operations.

## UPM Worked example

The ECD for event "Decision of Court" which was produced earlier will now be developed into an Update Process Model.

## UPM Worked Example: Group effects on ECD

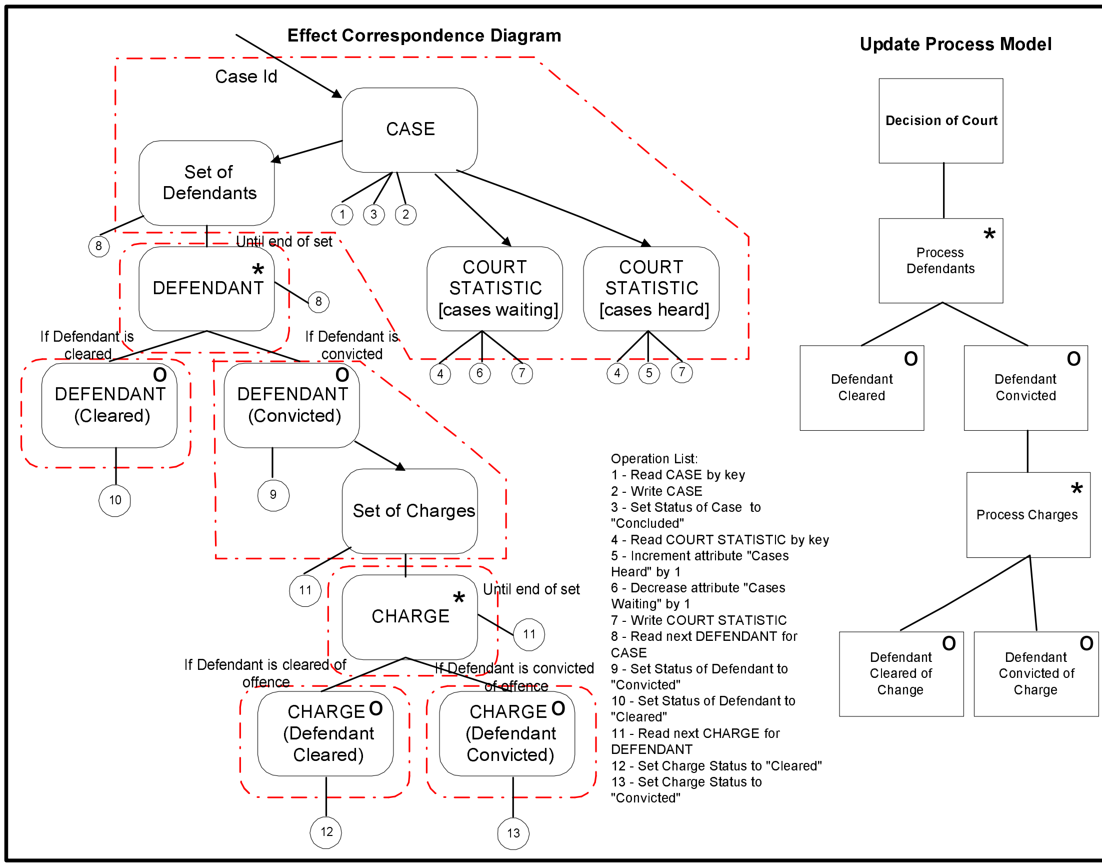
The grouping of effects/structure boxes in one-to-one correspondence on an ECD is shown below.


Case Id
CASE
Set of Defendants
1
3
2
8
Until end of set
DEFENDANT *
8
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
If Defendant is cleared
If Defendant is convicted
4
6
7
4
5
7
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
10
9
Set of Charges
11
CHARGE *
Until end of set
11
If Defendant is cleared of offence
If Defendant is convicted of offence
CHARGE (Defendant Cleared) O
CHARGE (Defendant Convicted) O
12
13

Operation List:
1 - Read CASE by key
2 - Write CASE
3 - Set Status of Case to "Concluded"
4 - Read COURT STATISTIC by
5 - Increment attribute "Cases Heard" by 1
6 - Decrease attribute "Cases Waiting" by 1
7 - Write COURT STATISTIC
8 - Read next DEFENDANT for CASE
9 - Set Status of Defendant to "Convicted"
10 - Set Status of Defendant to "Cleared"
11 - Read next CHARGE for DEFENDANT
12 - Set Charge Status to "Cleare
13 - Set Charge Status to "Convicted"

## UPM Worked example: Convert grouped effects into Jackson-like notation

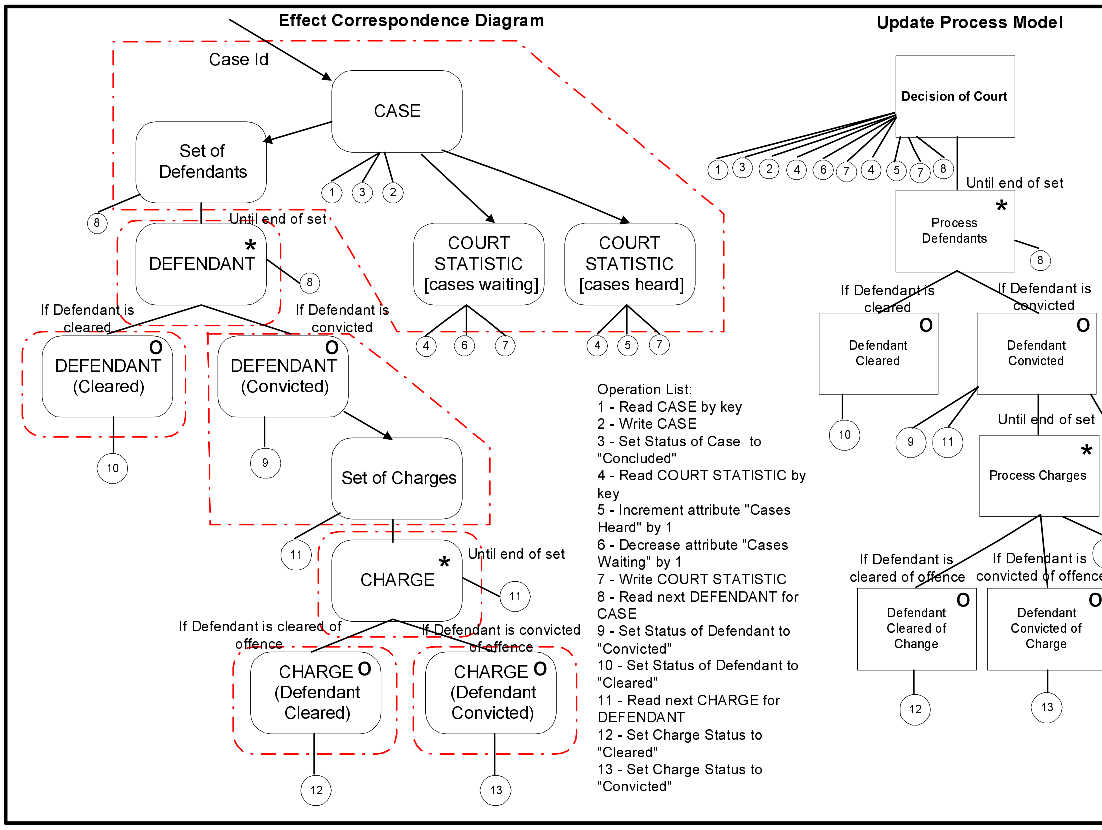
The grouped effects which need to be converted to a Jackson-like structure and the resultant structure is shown below.


Effect Correspondence Diagram
Case Id
CASE
Set of Defendants
Until end of set
DEFENDANT *
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
If Defendant is cleared
If Defendant is convicted
DEFENDANT (Cleared) O
DEFENDANT (Convicted) O
Set of Charges
Until end of set
CHARGE *
If Defendant is cleared of offence
If Defendant is convicted of offence
CHARGE (Defendant Cleared) O
CHARGE (Defendant Convicted) O
1
2
3
4
5
6
7
8
9
10
11
12
13
Update Process Model
Decision of Court
Process Defendants *
Defendant Cleared O
Defendant Convicted O
Process Charges *
Defendant Cleared of Change O
Defendant Convicted of Charge O

Operation List:
1 - Read CASE by key
2 - Write CASE
3 - Set Status of Case to "Concluded"
4 - Read COURT STATISTIC by key
5 - Increment attribute "Cases Heard" by 1
6 - Decrease attribute "Cases Waiting" by 1
7 - Write COURT STATISTIC
8 - Read next DEFENDANT for CASE
9 - Set Status of Defendant to "Convicted"
10 - Set Status of Defendant to "Cleared"
11 - Read next CHARGE for DEFENDANT
12 - Set Charge Status to "Cleared"
13 - Set Charge Status to "Convicted"

## UPM Worked example: Allocate operations and conditions to Update Process Model

The allocation of conditions and operations to an EPM is shown in the diagram below


Effect Correspondence Diagram
Update Process Model
Case Id
CASE
Set of Defendants
DEFENDANT
Until end of set
If Defendant is cleared
If Defendant is convicted
DEFENDANT (Cleared)
DEFENDANT (Convicted)
COURT STATISTIC [cases waiting]
COURT STATISTIC [cases heard]
Set of Charges
CHARGE
If Defendant is cleared of offence
If Defendant is convicted of offence
CHARGE (Defendant Cleared)
CHARGE (Defendant Convicted)
Decision of Court
Process Defendants
Defendant Cleared
Defendant Convicted
Process Charges
Defendant Cleared of Change
Defendant Convicted of Charge

Operation List:
1 - Read CASE by key
2 - Write CASE
3 - Set Status of Case to "Concluded"
4 - Read COURT STATISTIC by key
5 - Increment attribute "Cases Heard" by 1
6 - Decrease attribute "Cases Waiting" by 1
7 - Write COURT STATISTIC
8 - Read next DEFENDANT for CASE
9 - Set Status of Defendant to "Convicted"
10 - Set Status of Defendant to "Cleared"
11 - Read next CHARGE for DEFENDANT
12 - Set Charge Status to "Cleared"
13 - Set Charge Status to "Convicted"