Data Requirements Specifications from Data Flow Diagrams

Introduction:

In order to design a conceptual data model (as part of a software development project), the first step is to produce a data requirements specification (DRS).

Here, due to the limitations inherent in an educational environment, we must assume that a DRS is derived from a process model, rather than independently, by database designers, through a process of face-to-face interaction with stakeholders. In doing so, we assume that it is the systems analysts that have engaged in such interaction and that, as data analysts/database designers, we are operating downstream from the process modelling phase, in terms of the software development life cycle.

Thus, we assume that in order to produce a DRS, the database designers work from a process model and, if necessarily, elicit further clarification from the systems analysts who generated the latter from face-to-face contact with stakeholders.

These notes briefly introduce the notion of process models that are represented as data flow diagrams (DFDs) using the Gane-Sarson notation. Our goal here is to explain the syntax and semantics of DFDs with the specific purpose of allowing the reader, who is assumed to be a database designer, to extract from one such diagram a DRS, from which, in turn, a conceptual data model can later be derived.

Interpreting a Process Model:

A **process model** is an abstract representation of a system centred on its functions, also referred to as processes, and how they capture, transform, store and distribute the data among the components of the system, as well as across the latter's boundary and into the external environment.

A **data flow diagram** (DFD) depicts a process model in graphical form. A DFD is a graph with three types of nodes and one type of edge. The node types denote *functions*, *data stores* and *external entities* (also referred to as sources or sinks). External entities can be real agents (e.g., people, and organizations) in the real world or other processes that comprise a larger process of which the one described by the DFD is itself a component. Edges are directed and denote the *flow of data* from one node to another (possibly more than one in either case).

Nodes and edges are labelled. An edge label is a noun denoting the data item that is incoming or outgoing from a node. In the case of external entities, the label is a noun that indicates the agent, or system, that it represents. In the case of functions, the label is a verb phrase that indicates how it transforms the incoming edges into the outgoing edges. In the case of data stores, the label is a noun that indicates what data items flow into and out of the data store.

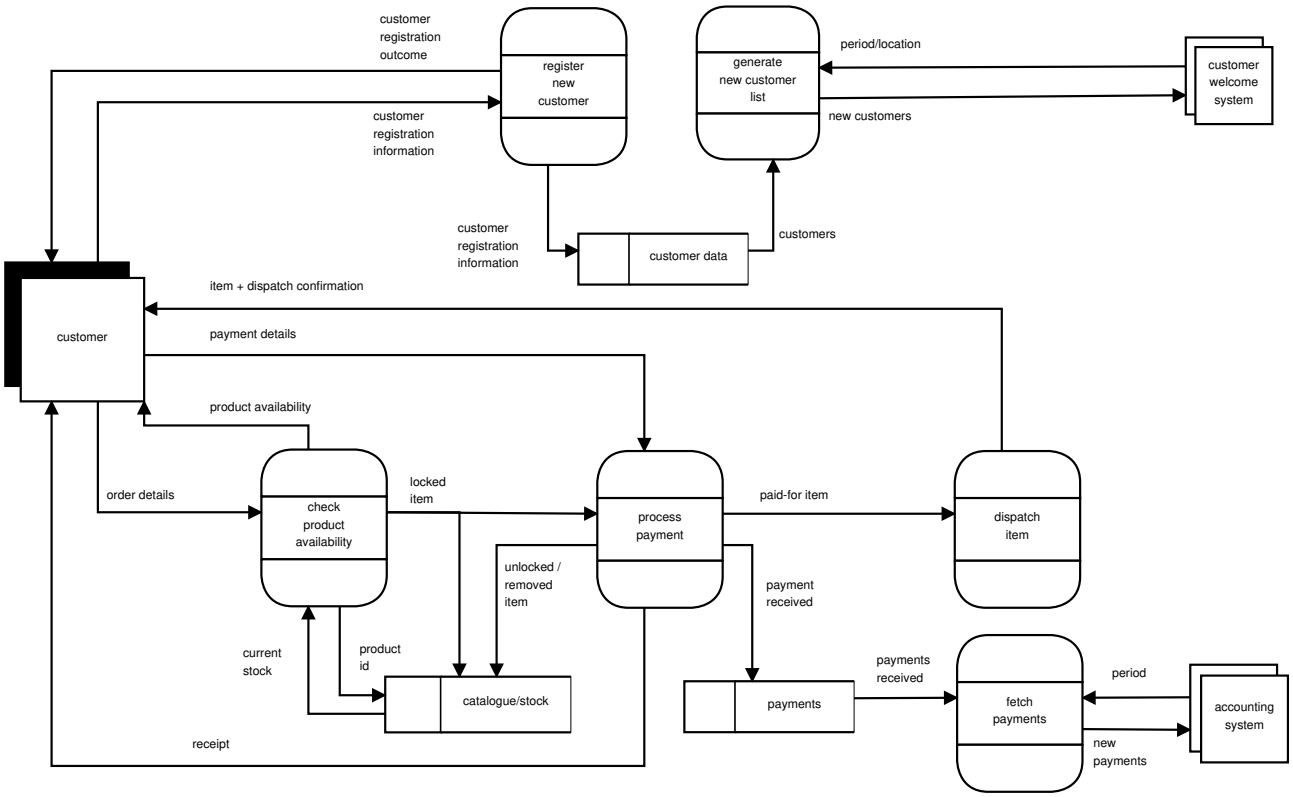

Figure 1: An Example DFD for (Part of) an E-Commerce System

Consider the example DFD in Figure 1, depicting a very small part of an e-commerce system.

In this example, one type of external entity is a customer. Another, is the customer welcome system, which, for example, sends a printed welcome pack containing information about terms and conditions, and so on.

In this example, at the top half of the DFD, there are two functions: one is to register new customers, the other is to generate a new customer list for the benefit of the customer welcome system.

The register new customer function has an incoming edge from the customer external entity labelled customer registration information and two outgoing edges: one, labelled customer registration information, into a data store labelled customer data and another, labelled customer registration outcome, into the customer external entity.

The generate new customer list function has two incoming edges and one outgoing edge. One incoming edge from the customer welcome system makes available the parameters that decide who goes into the list of new customers in terms of their location (say, Manchester) and the registration date (say, in the last calendar month). The other incoming edge, from the customer data store, indicates that the function (potentially) processes all existing customers whose information is held in the customer data store. The outgoing edge into the customer welcome system is the list of new customers in the requested period and location.

The bottom half of the DFD is more complex, as it covers the sale process itself, from an order to payment receipt and item dispatch. Make sure you can read the notation and understand the process being modelled, but be careful not to allow yourself to impose on it your own intuitions as what is the right or wrong way to do it. This is important in practice, but not here, not now.

The best intuition about DFDs is that they represent how data, stemming from external entities acting as sources, enters the process, is transformed by functions, possibly resting in data stores at different stages of the overall flows, and eventually leaves the process, destined to external entities acting as sinks.

Notice that the edges denote collections of data, not individual records, or data items. These collections can, potentially, be singletons, but are always thought of as collections. This implies that data store operations are assumed to be collection-based and that a function is conceived of as operating on an entire collection of items (i.e., it would not be inappropriate to think of functions as iterative control structures, or, possibly, recursive computations).

Notice that a data store represents data at rest, in the sense that the flow from one function to another is interrupted, or decoupled in time, by placing the data in the store.

Notice, finally, that a DFD has a logical boundary defined by its external entities. This means that, from the viewpoint of modelling the process that the DFD represents, we are unconcerned with that use was made of the inflowing data as well as what use is made of the outflowing data. So, we do not consider any interactions between external entities and, it also follows from the fact that they lie outside the process boundary that they are not allowed to act on stored data directly: there must be functions that mediate the interaction of external entities with the stored data.

One can draw DFDs at different levels of abstraction but the most crucial ones for software system development are those referred to as **level-0 DFDs**, which define how a particular system performs the major functions of an information system. So, a level-0 DFD defines the external entities that act as sources of data and how to capture data from them as well as also defining the functions that transform and distribute the resulting data products to external entities that consume them, possibly whilst requiring data to rest in data stores for particular purposes and reason (e.g., time decoupling, sharing, etc.).

Level-0 DFDs capture the primary functions in a process, i.e., those that are most essential to articulate the fundamental nature of the information system, and, in an organizational context, the strategy for adding value to the inputs and generate outputs that allow the organization to deliver what its stakeholders need and, hence, compete strongly in its target market.

It is important to bear in mind that DFDs are not meant to model timing, therefore it is best to draw, and interpret, DFDs as capturing a process that is in constant flow, and has never started nor will ever end.

For a DFD to be well-formed the constraints in Figure 2 must hold.

C1 Nodes and edges are identified by their name, so if a node or edge appears more than once in a DFD, typically to avoid graphical congestion and confusion, then it is the same node or edge. In the case of nodes, it is customary to draw a mark (e.g., a little diagonal line in the top-left corner) to indicate that this node also appears elsewhere.

C2 No function can have only inputs.

C3 No function can have only outputs.

C4 Data cannot directly flow from one data store to another, or from an external entity to another, or from an external entity into a data store, or from a data store to an external entity: only a function can move data, or for one function to that same function: in all such cases, an explicit function must mediate.

C5 A function transforms inputs into outputs, therefore the incoming edges must be different than the outgoing ones. If a function merely retrieves data (e.g., call it D) from a store S in order to distribute it to an external entity E without transforming, then we qualify, e.g., retrieved-D-from-S-for-E. Likewise, if a function merely obtains data from an external entity in order to pass it on to a function F or to store it in a store S, then we qualify, e.g., obtained-D-from-E-for-S (or F, if that is the case).

C6 A data flow is never bidirectional: two flows must be drawn and qualification is used to distinguish them.

C7 One data flow may fork into n flows, with the semantics the collection is identically replicated n times, one for which destination.

C8 n data flows may join into one flow, with the semantics that the different collections are merged in a non-specified manner (e.g., by set, or bag, union).

C9 A flow into a store has general update semantics (i.e., create, replace, update or delete).

C10 A flow out of a store has the semantics of retrieve (or fetch) without modification.

Figure 2: DFD Well-Formedness Constraints

Deriving a DRS from a Level-0 (New, Logical) DFS:

Recall that our specific purpose in these notes is to explain to the reader, who is assumed to be a database designer, how to derive a DRS from a DFD, from which, in turn, a conceptual data model can later be derived.

In software development, we often develop logical models and physical models, where the latter abstracts from the implementation details that is the purpose of the latter to capture. Also, for each of logical and physical models we often develop one model of the current system (if any) and another model of the new, desired system.

For the purpose in hand, i.e., deriving a DRS from a DFD, we will assume we have a logical level-0 DFD of the new, desired system.

What Does a DRS Capture?

Given a logical level-0 DFD of the new, desired system, the derivation of a DRS depends on obtaining answers to questions that elicit the necessary information that goes in a DRS. In order to motivate the questions, we now consider in more detail what a DRS captures.

A DRS aims to explicitly record (among other kinds of information of less relevance here) the information characterized by the questions in Figure 3.

How Does a DFD Help Elicit a DRS?

A data analyst would use the logical level-0 DFD of the new, desired system that has resulted from process modelling as an interface between her and the stakeholders to carry out the first step of data modelling, viz., eliciting a DRS.

In doing so, a data analyst would ask the kind of questions illustrated in Figure 4. Note that they are closely related to those in Figure 3 (though not all questions in the earlier figure are represented in the later one). However, the data analyst uses the logical level-0 DFD of the new, desired system to phrase them in terms that are closer to the way the stakeholders view the workings of their organization.

A Concrete Example: Using the DFD to Ask the Right Questions.

Now, consider again the simple e-commerce system in Figure 1. Example questions that a data analyst may be prompted to ask when we're guided by that level-0 DFD and her common-sense intuitions are shown in Figure 5. These are keyed to the question types in Figure 4, but, note that, here, we don't have a real stakeholder to interview, so we'll assume some likely answers based on common knowledge, for the sake of practicing formulating these kinds of questions. Note, also, that the examples below are not exhaustive, the example questions do not systematically traverse all of the DFD in Figure 1: in real software development, you would do so, but then, the interaction and reflection processes that go into the generation of a complete DRS typically take in the order of weeks or months.

A Concrete Example: Transforming the Answers into a DRS.

Figure 6 illustrates how a data analyst would use the answers obtained from the stakeholder to derive a DRS for the new system. This DRS, in turn, would be the basis for designing the database to support the new system, as we shall explore later in this course unit.

I1 What entity types (or classes of objects) will have data about them stored?

I2 What subtypes or supertypes or composite types or union types of entities of interest are of interest themselves?

I3 What relationship types there exist between the entity types of interest?

I4 Is the participation of every entity of a certain type in a given relationship type mandatory or optional (i.e., can, or cannot, there be an entity in the store that does not participate in a relationship of that type)?

I5 When an entity of a certain type participates in a given relationship type, does it do so once only or multiple times?

I6 What attributes characterize the data that is stored about entities or relationships of a certain type?

I7 Is any of an entity's attributes identifying and, if so, which?

I8 Is the scope of identification of an identifying attribute global (i.e., unique with global scope) or partial (i.e., dependent on the identifying attribute of some other entity, meaning that it needs the latter to become globally unique)?

I9 Is any attribute is multivalued?

I10 Is any attribute computable from other attributes?

I11 Is any attribute complex, i.e., composed of other, component, attributes?

I12 What is the type of each attribute and what rules, if any, constrain the values it can take?

I13 What constraints, if any, characterize the valid states of the stored data and hence must be satisfied at all times?

I14 Does the history of any data item need to be recorded?

Figure 3: The Information Content of a DRS as a Set of Questions

Practice Tasks:

Consider the example level-0 DFD in Figure 7, depicting a simple web store front-end system.

1. Using Figure 7 as your reference, and Figure 5 as an example to guide you, write down one or more questions of each of the types (i.e.,**Q1** to **Q8**) in Figure 4.

2. Using Figure 5 as an example to guide you, provide common-sense, intuitive answers (that conform with Figure 7) for the questions you wrote down in response to the previous task.

3. Using Figure 6 as an example to guide you, write down the DRS items that follow from the answers you gave in response to the previous task.

Q1 For each data flow, what is inside it? Information about a stakeholder (e.g., an account holder)? Is it perhaps a product (e.g., insurance) or service (e.g., currency exchange) bought or sold? Or is it a document (e.g., the record of a transaction in an ATM)? Must this information be kept stored? Or is it used and discarded?

Q2 For each identified stakeholders, products, services, documents, etc., does it have supersets or subsets that we take an interest on? For example, the various products (i.e., current accounts, savings accounts, insurance, investment funds, etc.) can be grouped as types of account since they share some common properties (e.g., an opening date, a balance, etc.) while having specific properties of their own (e.g., current accounts have an interest-due rate for overdrafts, savings accounts have an interest-paid rate, etc.). As an example of subsetting, some investment accounts are on stocks, others on commodities, and so on.

Q3 Do the identified stakeholders, products, services, documents, etc., have a unique, strong, global identifying property? For example, in the UK, a person typically has a National Insurance number that is a unique, strong, global identifying property of that person. In contrast, an address is only possibly unique if one takes the postcode and adds the house number. Also, a flat number is unique only in the context of a given block, i.e., it's not a strongly, globally identifying, it's only weakly, partially so because it must be concatenated with the identifying property of the building before we be certain which flat is being referred to.

Q4 For each stakeholder, product, service, document, etc., what information about it is flowing (e.g., for a service like currency exchange, these could be the source currency, the target currency, the exchange rate, the amount, the date and time, the source and target accounts, etc.)?

Q5 Which, if any, of these properties composite (i.e., made of component properties, like an address might be composed of house number, street name, and flat number)? Which, if any, is multivalued (e.g., a block of flats may have several entrances, e.g., front, left, right, back)? Which, if any, is derivable from others (e.g., the total insurable value of a building is the sum of the insurable values of each flat in it plus the insurable value of the common areas)?

Q6 For the identified stakeholders, products, services, documents, etc., how do they relate to each other? For example, an account holder contracts insurance.

Q7 For each such identified relationship type, must it always hold for each stakeholders, products, services, documents, etc.? For example, is it the case that every account holder must have contracted an insurance product, meaning that unless some insurance is contracted accounts cannot be held with the company, or is it instead the case that someone can hold an account without contracting any insurance? In the other direction, is it the case that every insurance product must have some account holder that has contracted it, or is it the case that there can be insurance products that have no contractors yet?

Q8 For each identified relationship type, can my stakeholders, products, services, documents, etc. participate in it? For example, is it the case that an account holder can contract many insurance products or just one? In the other direction, is it the case that an insurance product can be contracted by many account holders or exclusively by one account holder only?

Figure 4: Some of the Questions a Data Analyst Asks so as to Derive a DRS

Q1 Must information about customer external entities be stored?
A: Yes, we store data about our customers.

Q2 Is the customer data data store just a temporary resting place in the flow of data or does it really constitute a data asset of the organization?
A: It is a data asset for us.

Q3 Are there different types of customer? For example, premium and freemium?
A: No, there aren't different subsets of interest among our customers.

Q4 Is there some piece of information that distinguishes one customer from all the others?
A: Yes, when a customer registers, we assign it a customer id number whose uniqueness we enforce rigorously.

Q5 What data do you want to hold about a customer?
A: Besides the customer id, the password, the name, the address, the date of birth, the age, the phone numbers.

Q6 Are any of these composite, i.e., made of parts?
A: Yes, we break down the name into first name and last name. Also, we break down address into postcode, house number, and flat number.

Q7 Do I take it that you store more than one phone number for a customer?
A: Yes, we do.

Q8 And do you differentiate between, say, landline and mobile numbers?
A: No, we don't.

Q9 Am I right in assuming that we can compute the age of the customer on any given date using the date of birth?
A: Yes, I suppose you are right.

Q10 How would you describe the sales process?
A: A customer makes an order. The order list the products of interest, which are checked for availability. If the products are available, the customer sends us payment details. These are then processed, so that a receipt reaches the customer. Paid-for items are then dispatched to the customer with a dispatch confirmation.

Can I confirm the relationships and/or documents I have identified? The relationships are: customer makes order, order lists product, customer sends payment, product is prepared for dispatch, customer receives product.
A: This seems right, in a nutshell.

Am I right in assuming that, therefore, over and above data about customers, you want to hold data about products in a catalogue, orders and payments received and items dispatched? *A: Yes, this is correct.*

Q11 Now, going back to the customer makes order and the order lists product relationships. Am I right in assuming that a customer can have many orders in your data stores but each order can only refer to a single customer, and also that a order can list many products and that each product can be listed in many orders?
A: Almost all of it is right. The unusual detail is that we actually allow an order to refer to more than one customer.

Q12 Now, let stay with the customer makes order and the order lists product relationships. Am I right in assuming that customer data is kept even if the customer may not have made any order, but that an order must have been made my some customer, and also that a order must list at least one product and but that product data is kept even though it may not have been listed in any order?
A: Yes, you got all of it right this time.

Figure 5: Data Analyst Questions Guided by Figure 1

R1 There is a need to store data about each customer.

R2 There are no sub- or supersets of interest for customer.

R3 The identifying property of customer is a company-assigned customer id.

R4 Besides the customer id, a customer has the following data about it stored: password, name [composite of first name and last name], address [composite of postcode, house number, flat number], date of birth, age [derived from date of birth, the phone numbers [multivalued].

R5 *Here, by analogy with customer, there would be data requirements specs for order, product, payment, dispatch. In what follows, we assume they exist.*

R6 A customer makes an order. A customer can make many orders but there may be a customer that has not made any orders. An order can be made by many customers and there may not be an order that has not been made by some customer.

R7 An order lists a product. An order can list many products and a product can be listed in many orders. An order must list a product but a product may not have been listed in any order.

R8 *Here, all the other relationships would have been considered.*

R9 *Also, the data analyst would consider whether more needs to be recorded about each relationship, e.g., the date an order was made, the credit/debit card number used to make a payment, etc.).*

R10 *Finally, we're omitting here the many constraints that data needs to obey, from valid values (e.g., of dates, or postcodes, etc.) to business rules (e.g., that there is a day-limit on the total amount charged of a single credit/debit card number).*

Figure 6: (Partial) Data Requirements Specification from Figure 1

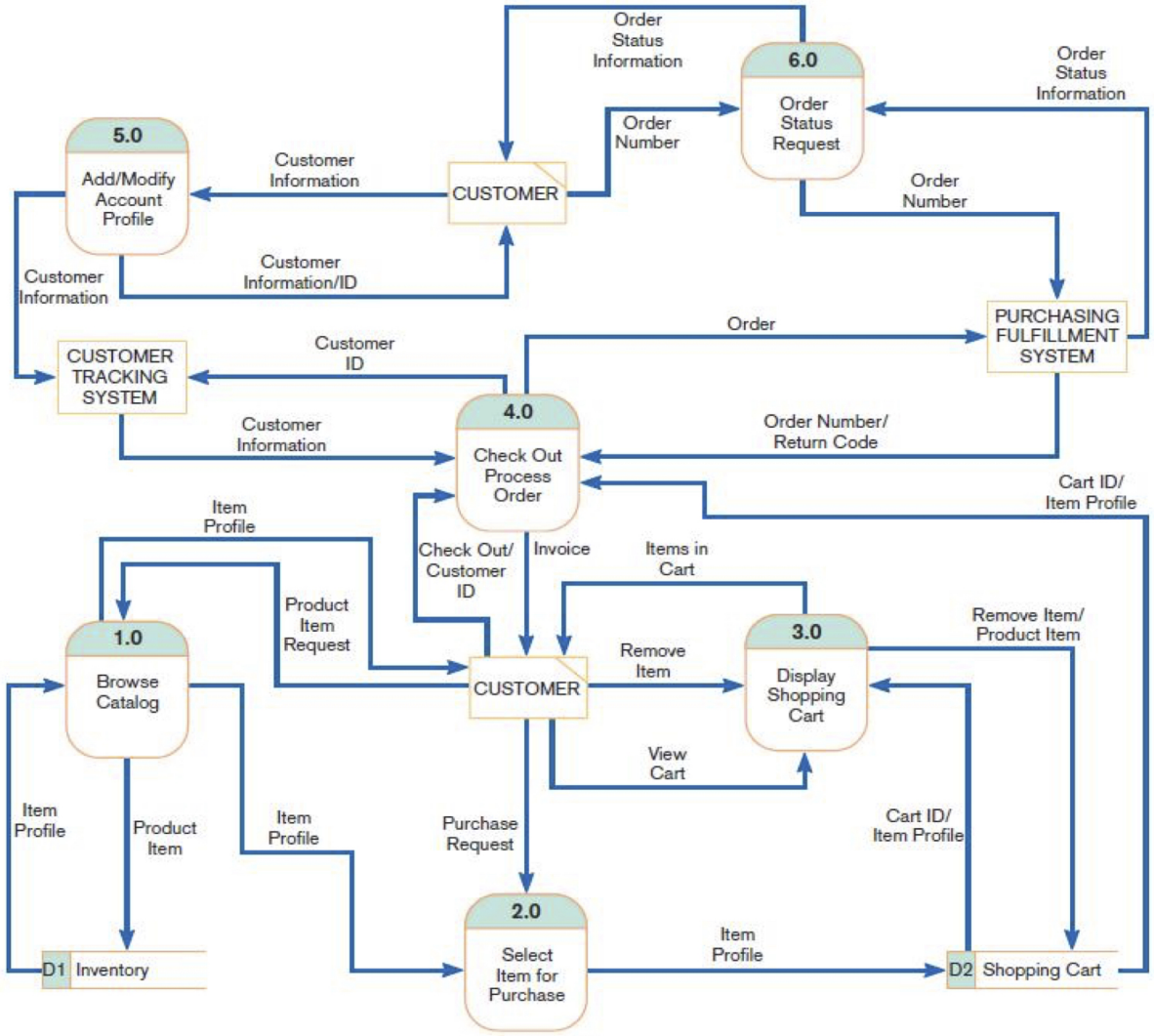

Figure 7: A Level-0 DFD for a Web Store