Requirements Patterns Via Events/Use Cases

Suzanne Robertson
The Atlantic Systems Guild Ltd
11 St Mary's Terrace
London W2 1SU
Phone: 0171 262 3395
email:100065.2304@compuserve.com

Abstract
This paper illustrates how event/use case modelling can be used to identify, define and access requirements patterns. Event/use case partitioning is widely used as a way of breaking a system into manageable, business-related chunks. Each event/use case is then the subject of detailed analysis, design and implementation. This paper illustrates how event/use case modelling can also be used as the basis for identifying and defining requirements process and data patterns. Given a book of requirements patterns, each project uses context analysis and event/use case partitioning as a way of accessing relevant patterns.

Note 1: the terms event (reference McMenamin and Palmer '84) and "use case" (Jacobson '92) are used to mean the same thing. I acknowledge that there are some differences in application and notation, but for the purpose of identifying and using process patterns the models are interchangeable.

Note 2: all references to "system" refer to all the system's processes (computerised, manual, mechanised) not just the processes that are computerised.

Keywords: requirements pattern, event, use case, requirements analysis.

What is a Requirements Process Pattern?

A requirements process pattern captures the processing policy that responds to a specific business event or use case. Each process pattern is bounded by its own input, output and stored data and can be treated as a mini system. Requirements process patterns are similar to the building patterns specified in Christopher Alexander's Pattern Language (Alexander '77). Requirements process patterns are used at the start of a project to help specify user requirements. Of course, in order to be able to use the patterns they must first be discovered and defined. The discovery is done by making abstractions from event-response or use case models. The definition is done by using a combination of modelling techniques designed to facilitate capture and avoid repeating the work necessary to analyse, define and review requirements that have already been specified .

An Example

Suppose that you are working on a system for a book shop. One of the events/use cases within your context is: *Customer wants to buy a book.* Figure 1 is a summary of the system's processing in response to the event. When the Customer places an *Order For Book* the system responds with some combination of: *Refused Book Order, Out Of Print Book, Back Order Notification, Invoice For Book + Physical Book.*

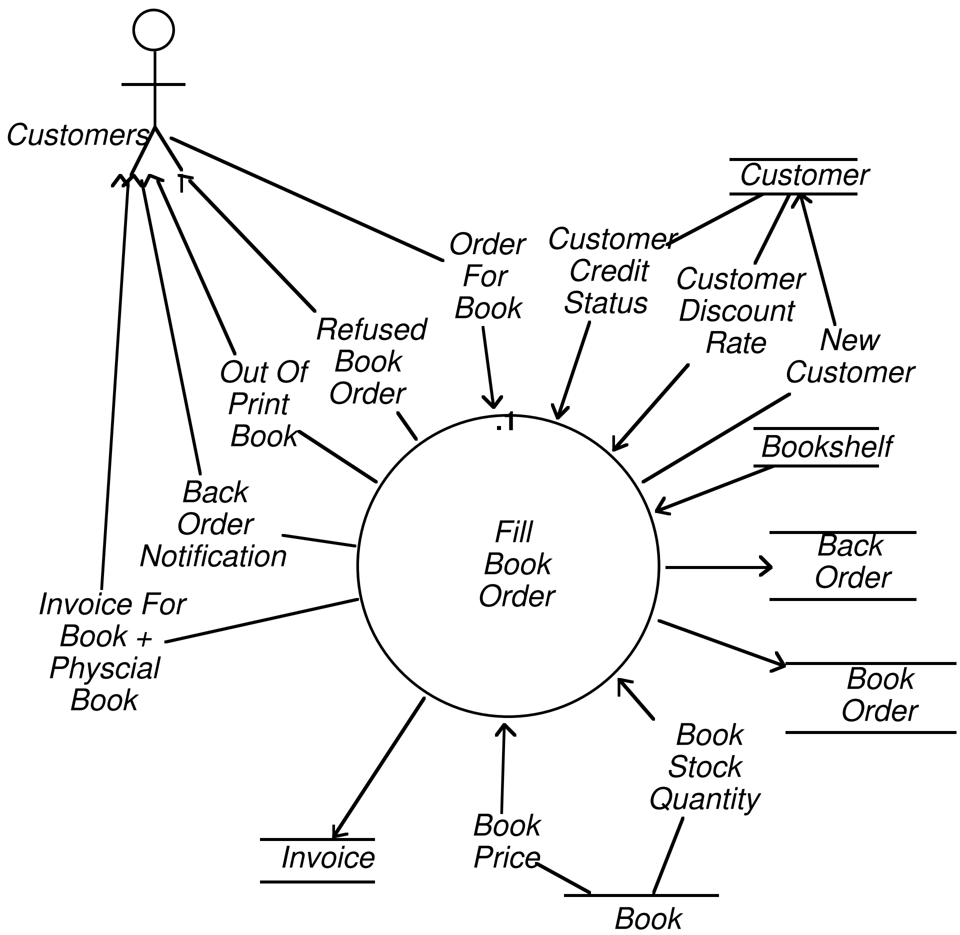


Figure 1: A summary of the system's response to the event: Customer wants to buy a book.

Notice that in order to carry out its processing policy in response to this event, the system needs to have access to stored information about: *Customer* and *Book* and the system records some information about *Back Order* and *Invoice*. The system also needs access to the physical books on the *Bookshelf*.

In order to understand more details of the system's processing policy we can trace the event response/use case through the system and identify the individual processes. The more detailed event-response process model in Figure 2 shows us that the *Order For Book* is input to the *Check Customer Credit* process. The process uses the stored *Customer Credit Status* to produce either a *Refused Book Order*, which goes back to the Customer, or *Approved Book Order*, which is input to another process in the system. The event-response continues to trigger processes within the system until all of the ripple effects either stop at a store of data or return to the customer.

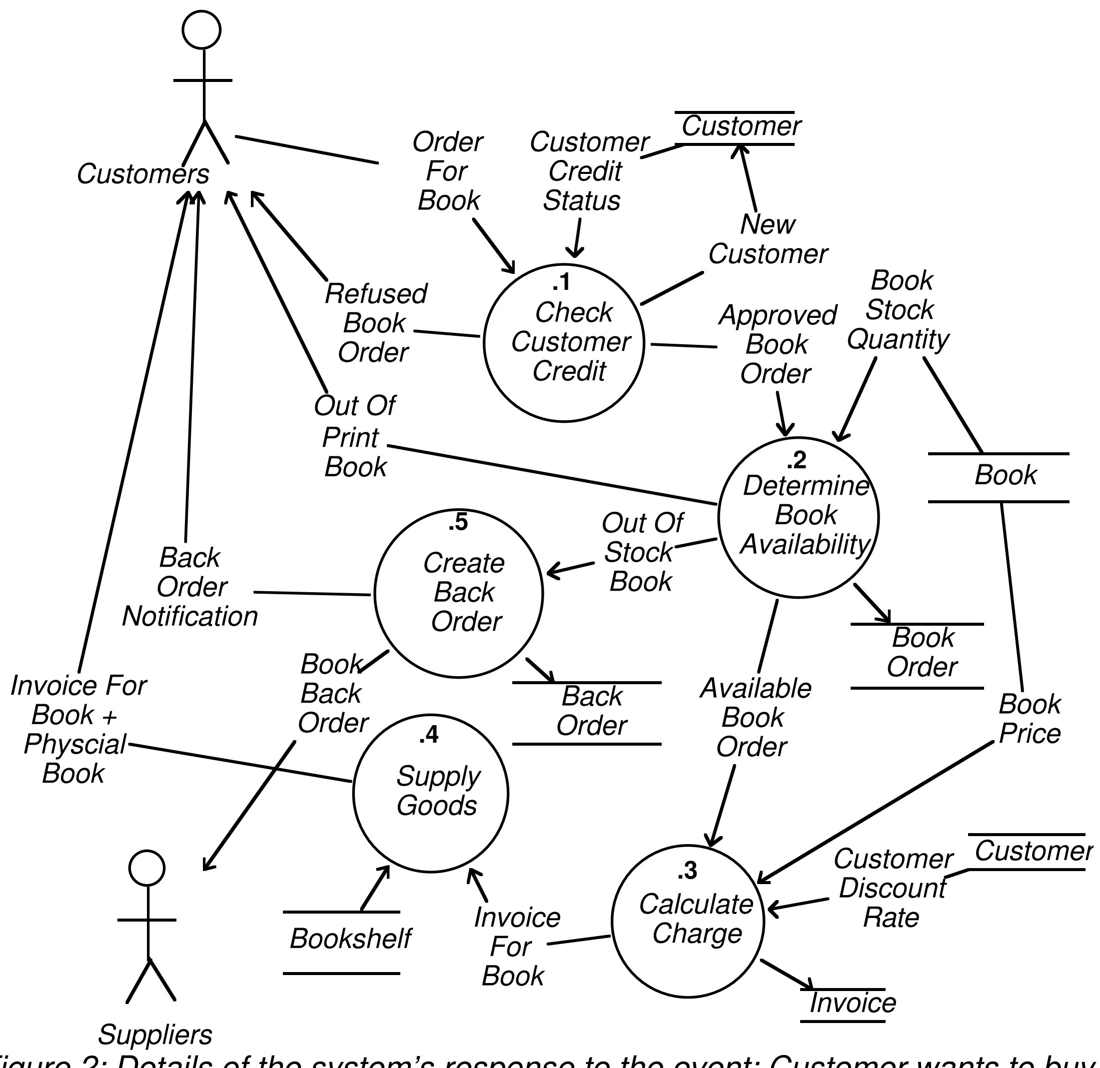


Figure 2: Details of the system's response to the event: Customer wants to buy a book.

The model is made more precise by defining all of the terms that are used and by specifying the detailed processing policy for each process. For example, *Order For Book* could be defined as a flow of data containing:

Customer Name
Customer Address
Book ISBN Number
Book Title
Book Author

Similarly, *Customer* could be defined as a store containing the following information for each *Customer:*

Customer Name
Customer Address
Customer Credit Status
Customer Discount Rate

The definition process continues until each elemental term is defined:

Customer Name is the identifier for a Customer

Customer Credit Status is Payments Up To Date or Payment Overdue

The processing policy for Check Customer Credit could be specified as:

Look for the stored *Customer Credit Status* that corresponds to the name and address in the *Order For Book*
If a match is found
    If *Customer Credit Status* shows payments are up to date
        then produce an *Approved Book Order*
    else (credit is not OK)
        produce a *Refused Book Order*
else (a match is not found)
    record a *New Customer*
    produce an *Approved Book Order*

The processes, interfaces and stored data components of this event-response/use case model are a specification for how this particular system responds to the event: *Customer wants to buy a book.* Now suppose that we have built other event/use case models for similar events in other systems. Then we could use those event/use case models as the raw material for discovering a reusable requirements process pattern.

Similarities and Differences

Let's look at another event response/use case before considering how we can discover more useful generic requirements processing patterns. Figure 3 is an event response/use case model for the event: *Customer wants to buy a service.*

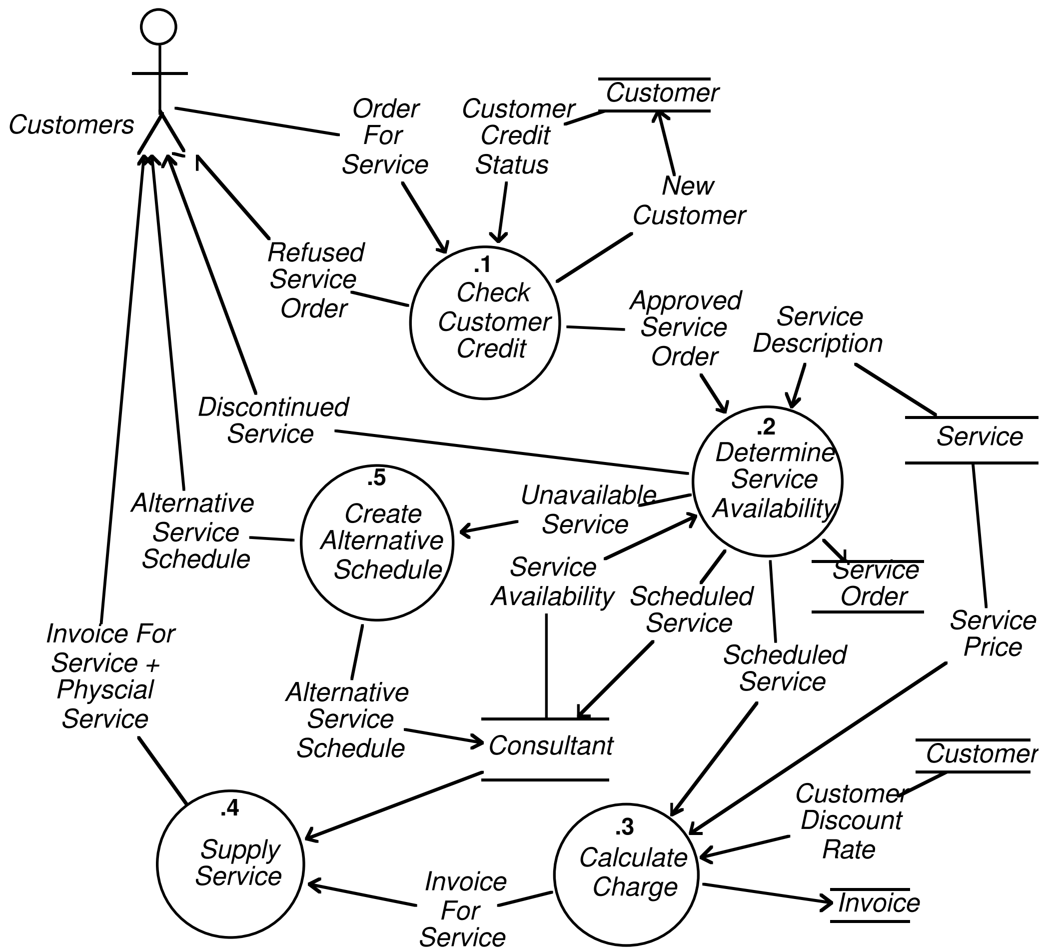
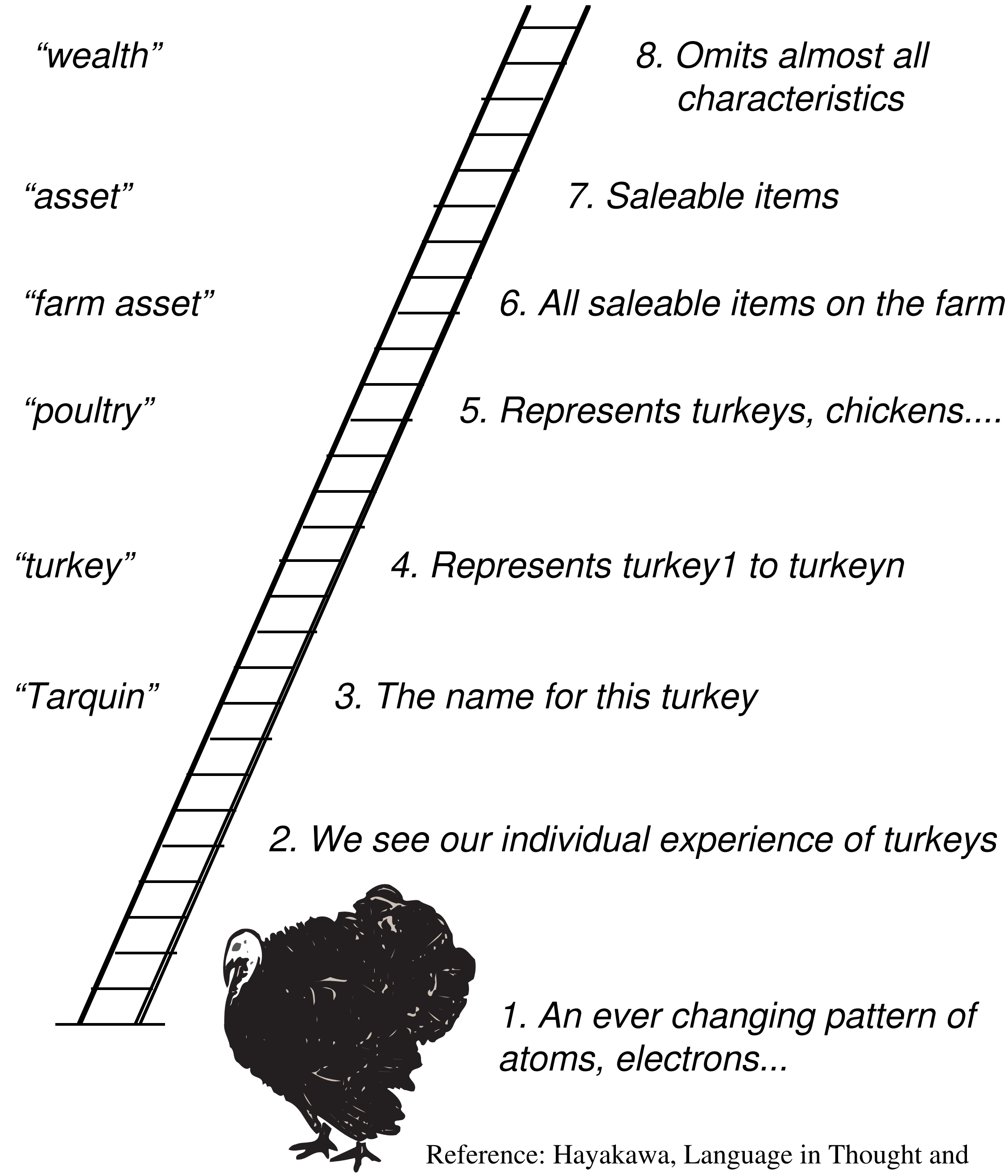


Figure 3: The system's response to the event: Customer wants to buy a consulting service.

This system is dealing with the supply of consulting services. At first glance the supply of consulting services has very little in common with our previous example which is filling book orders. However we can discover many similarities by making abstractions. In the first example the subject matter is concerned with supplying books to customers. In the second example we are supplying services to customers. So look at the two process models and ask the questions: what similarities are there between books and consulting services? Your answer might be something along these lines:

Making Useful Abstractions

Within the context of these event responses/use cases, both books and consulting services have descriptions, prices and availability. We use the abstraction ladder in Figure 4 to identify common characteristics and to define a higher level of abstraction. Both books and consulting services are *products* that we sell in response to orders from customers. Our processes for supplying books and services have some similarities, for instance we check the customer credit and calculate the charge in both cases. So we can say we have some processing policy that is common to all products. Of course there are also some differences between the two event responses because of the individual differences between books and services.



Reference: Hayakawa, Language in Thought and Action, George Allen & Unwin, London 1970

Figure 4: The abstraction ladder illustrates how discovery of generic patterns requires thinking at different levels of abstraction

Determine Book Availability checks to see if the book is in stock and if it is then the book is available to fill the order. If the book is out of stock then, providing it is still in print, the *Create Back Order* process creates a back order for the book. *Determine Service Availability* checks to see if there is a consultant available to provide the service when the customer wants it and if there is then the consultant is scheduled. If there is no consultant available then *Create Alternative Schedule* produces an alternative schedule which is offered to the customer.

By abstracting from the above description we can build an event response/use case model that abstracts the similarities and specifies the differences between the two sample events.

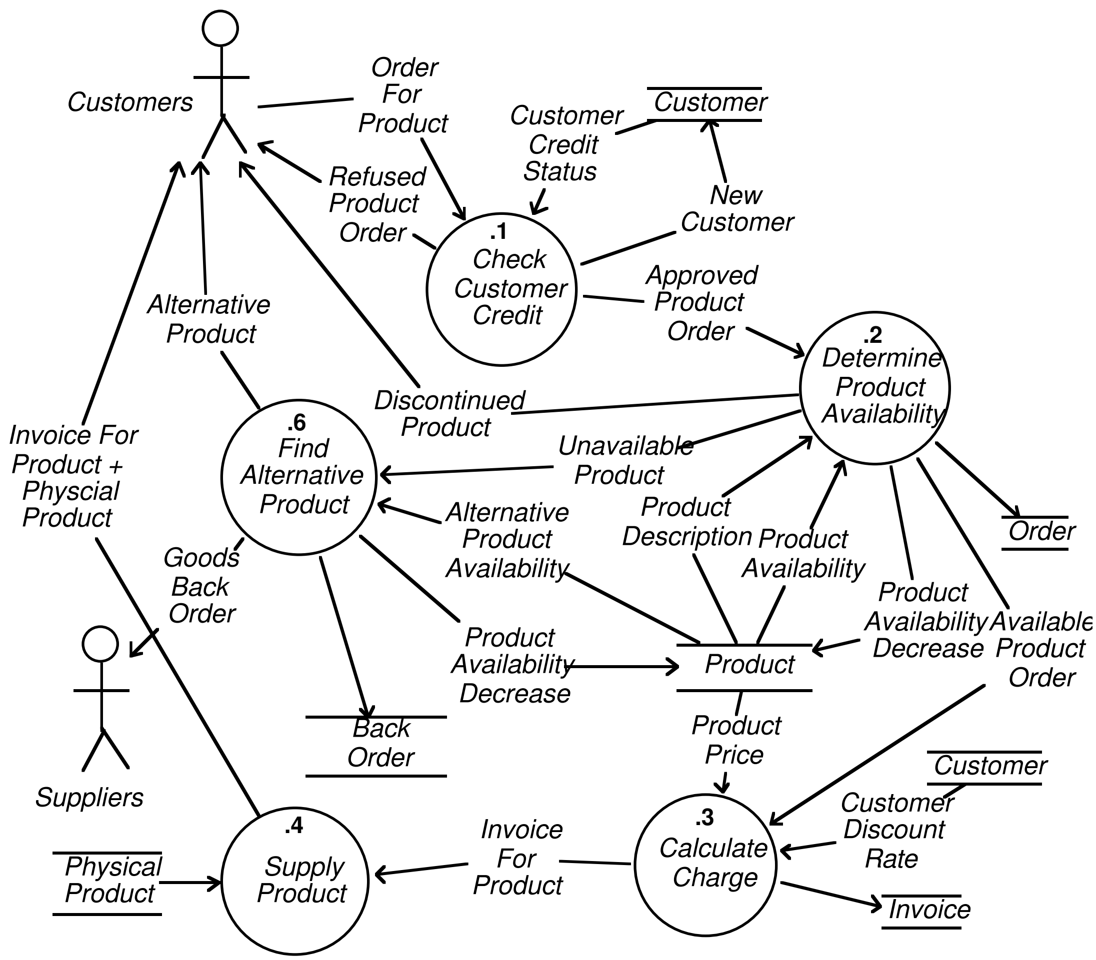


Figure 5: A requirements process pattern for the event: Customer wants to buy a product.

We discover that some parts of the event response have the same pattern regardless of whether we have a goods product or a service product. *Check Customer Credit* behaves the same way for a book order, a stationery order, a consultancy order or an order to fix the plumbing. Other parts of the event response vary depending on the type of product with which they are concerned. For instance, *Find Alternative Product* produces a back order in the case of a goods-related product. However a service-related product requires a response that looks for alternative ways of scheduling the service that the customer has requested. These variations in response depending on product type can be specified as alternatives within the pattern's description of the individual process.

To become good at making useful abstractions you need to be able to classify the same subject matter at different levels of importance. This is easy to say, but is difficult to do. If you know a lot about a subject a natural tendency is to think of the world in terms of that subject. This tendency makes it difficult to see similarities between seemingly disparate subjects. However, if you are building event/use case models in a consistent way, you can use this consistency to help you to make useful abstractions by looking for similarities between models. Some characteristics to consider are matching of:

- Event/ use case names
- Process names
- Process input and output data flow names
- Names of stores
- Names of dataflow and store contents
- Value ranges of dataflow and store contents
- Names of store contents
- Process policy. The more formally this is stated eg. predicate logic, the easier it is to match similar policy

The more consistency you employ in your approach to specifying events/use cases the more potential exists for automated identification of similar characteristics.

The point of making the abstraction is that the requirements process pattern could be used as the starting point for any system that receives orders from customers. The pattern makes it possible to use detailed knowledge that has been analysed, specified and refined by many other requirements engineers.

The Connection With Data Patterns

The patterns we have reviewed all focus on the system's processing response to one event or use case. These patterns are referred to as requirements process patterns. In each requirements process pattern we see that individual processes are either referencing or creating stored information. In order to help understand the complexity of the system, we can also use the concept of events/use cases to capture requirements data patterns.

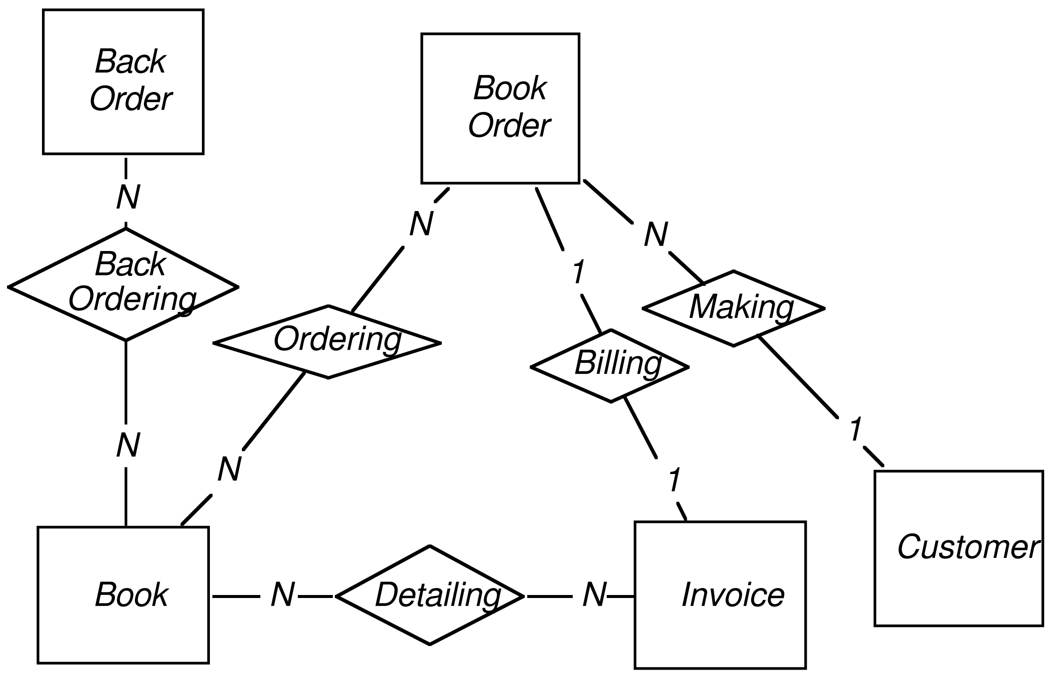


Figure 6: A requirements data model for the event: Customer wants to buy a book.

Rather than building one gigantic, unmanageable data model, we can use event/use case partitioning and build a data model to support one event/use case. This is similar to the approach we take when we build individual process models for each event/use case.

The event response/use case model in Figure 1 shows the processing that takes place in response to the event: *Customer wants to buy a book*. Figure 6 is the data model for the same event. This model specifies the business entities and relationships that must exist in order for the system to respond to the event. For example the model shows us that a *Book* has an *Ordering* Relationship with *Book Order*. The model also tells us that an instance of a *Book* can participate in an *Ordering* relationship with N (many) *Book Orders* and that a given instance of a *Book Order* can participate in an *Ordering* relationship with many *Books*.

Figure 7 is another example of using the event/use case partitioning to build a data model that focuses on the data for one event: *Customer wants to buy a consulting service*

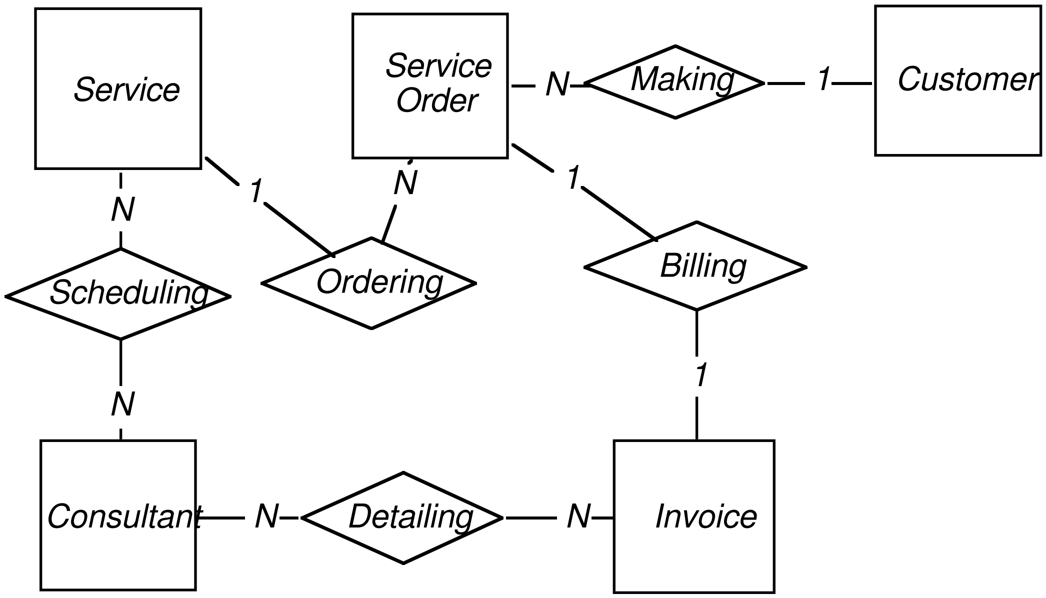


Figure 7: A requirements data model for the event: Customer wants to buy a consulting service.

The data models are a static view of the entities and relationships. The process models contain details of the creation, referencing, updating and deletion of the entities and relationships. In Figure 1 we see that process 1: *Check Customer Credit* references a store of the *Customer* entity to find out the *Customer Credit Status*. We can also see that under some circumstances, described in the process specification, the process creates a *New Customer.* Other processes that have their own reasons to being interested in the *Customer* entity will specify those details in their individual process specification.

The process model contains definitions of the stores that are accessed by processes.

e.g.. *Customer =*
Customer Name
Customer Address
Customer Credit Status)

The store *Customer* on the process model is equivalent to the entity *Customer* on the data model, so the same definition supports both the process and data models.

The more we know about the details of an event response, the better we are able to specify a more abstract pattern that will be helpful in a variety of circumstances. The data model in Figure 8 is the result of abstracting the data models for *Customer wants to buy book* and *Customer wants to buy consulting service*.

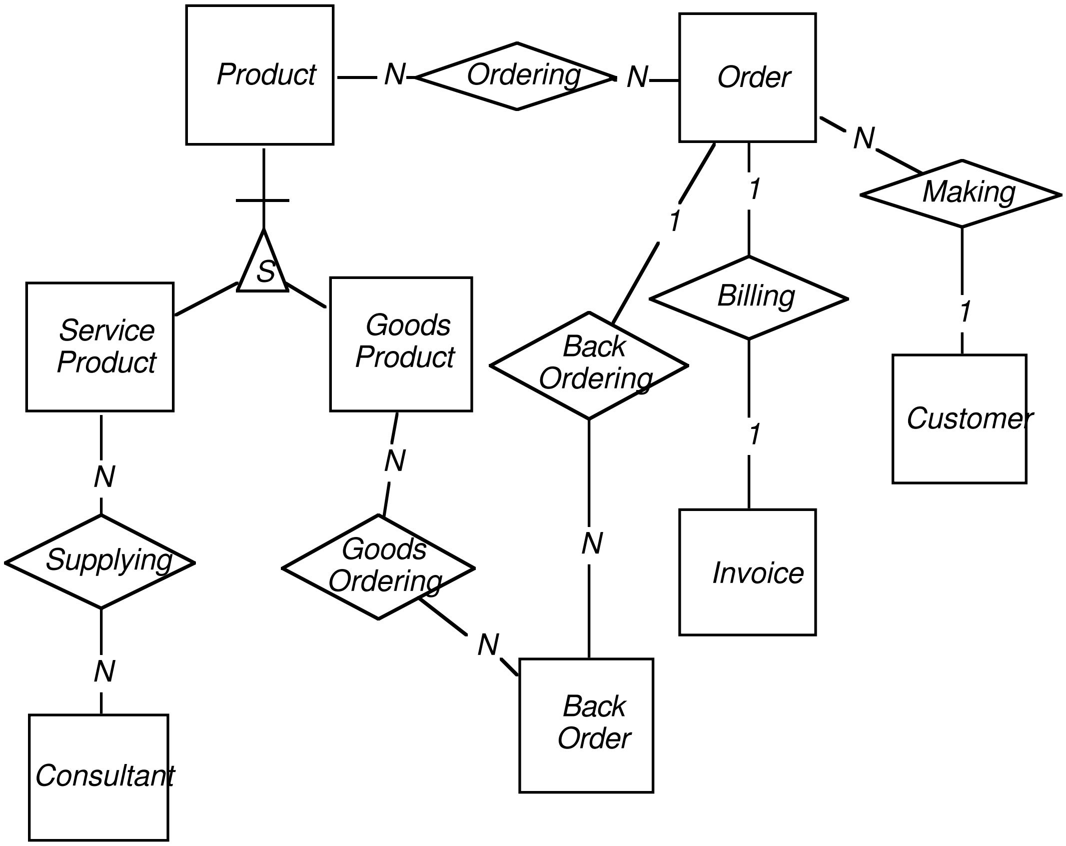


Figure 8: A requirements data pattern for the generic event: Customer wants to buy a product.

The model shows us that a *Service Product* and a *Goods Product* share the characteristics defined in *Product*, and that all *Products* can participate in the *Ordering* relationship with *Order*. However the *Service Product* and the *Goods Product* have their own unique relationships. We could use this model as a pattern for any event/use case when a customer wants to buy any type of product.

Storing Your Requirements Process Patterns

To make your patterns accessible you store them in a Pattern Book using a consistent format. A template for storing your patterns is:

Pattern Name: A descriptive name to make the pattern part of your vocabulary

Context: The boundaries within which the pattern is relevant

Solution: A description of the pattern using a mixture of words, graphics and references to other documents

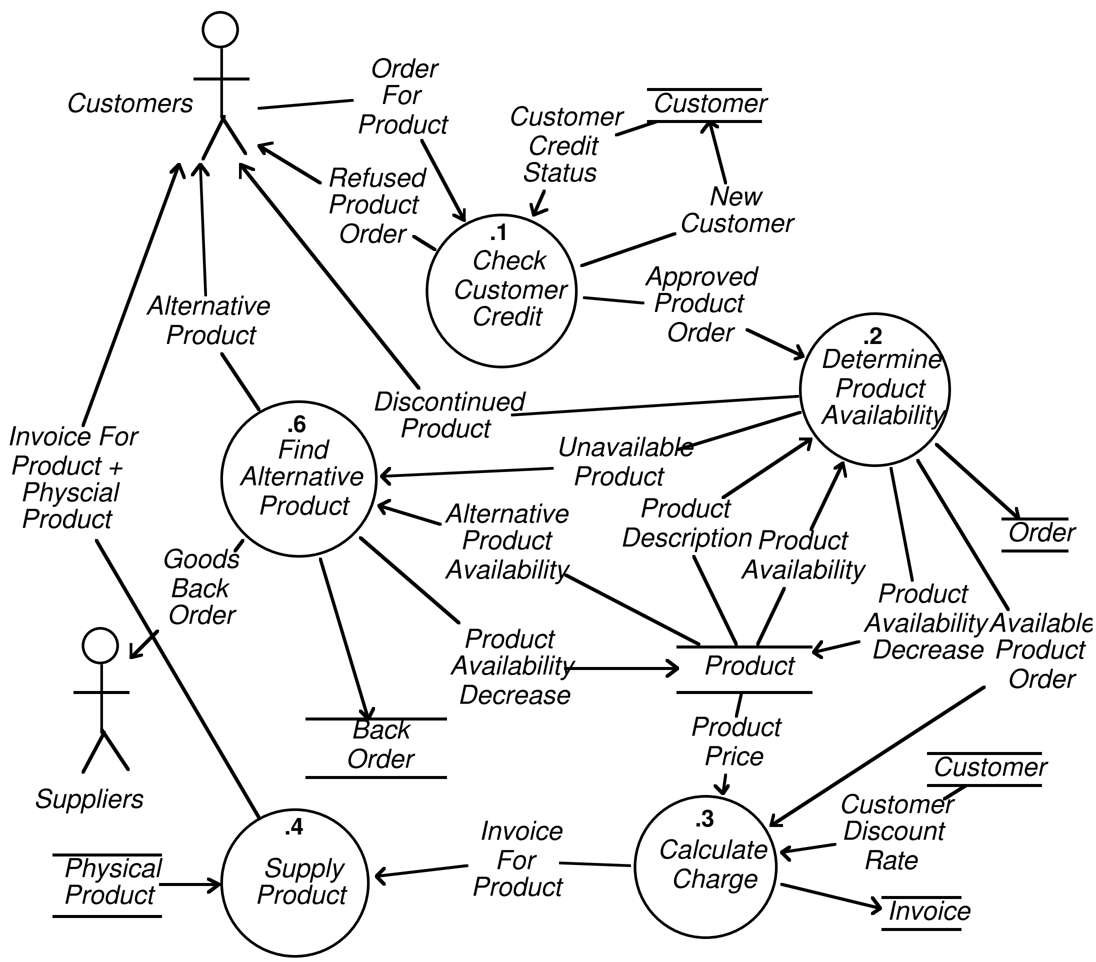
Related Patterns: Other patterns that might apply in conjunction with this one. Other patterns that might help to understand this one

Here is an example of using the template to record the requirements process pattern for *Customer wants to buy a product*

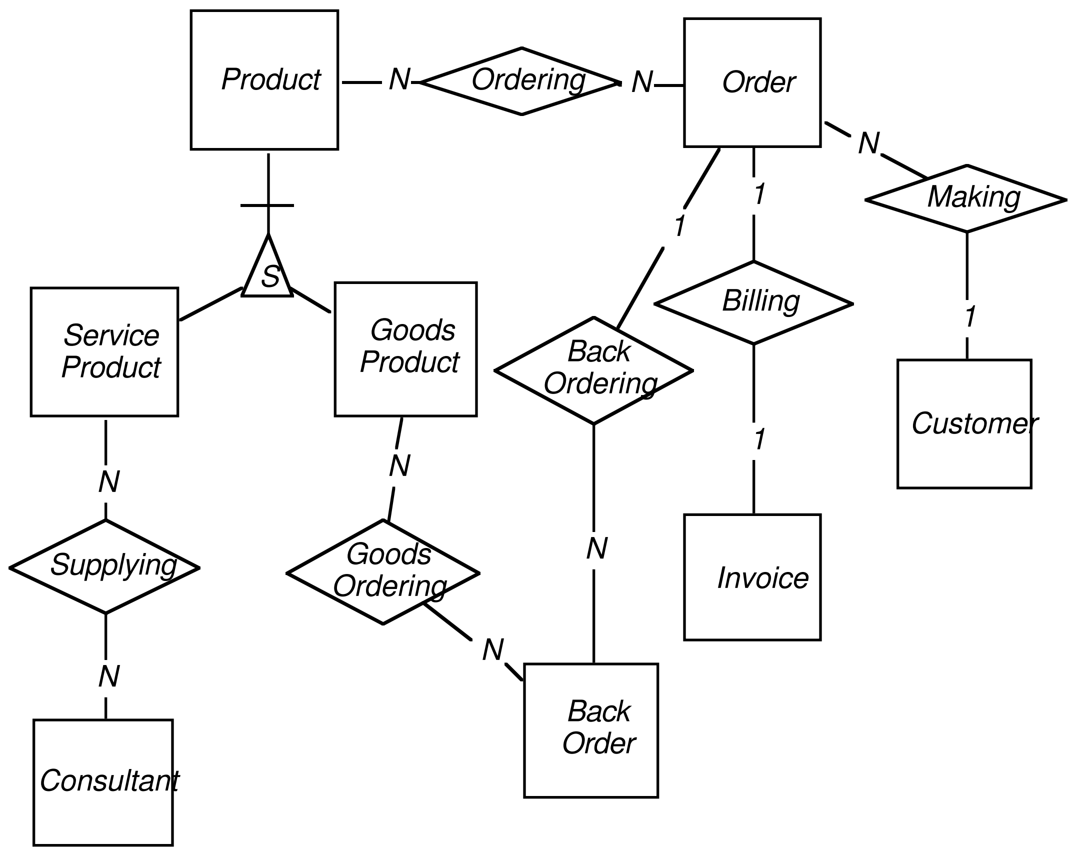
Pattern Name: Customer wants to buy a product

Context: A pattern for receiving product orders from customers, supplying or back ordering the product and invoicing for the product.

Solution: This event/use case process model defines the pieces of the pattern. Each actor, process, data flow and data store is defined in detail in the attached text using the same names as are used in this model:



This event/use case data model defines the entities and relationships within the context of this event:



*Note: for reasons of space I have not included the data and process definitions.

Related Patterns: Customer cancels back order, Customer makes payment, Customer returns faulty product, Supplier delivers goods.

Selecting and Using Process Patterns

Suppose that you have access to a pattern book containing a variety of requirements process patterns. At the beginning of a project your first concern is to identify any existing patterns that are relevant to your work. If your patterns are catalogued by event/use case, then you can use the event as your accessing mechanism.

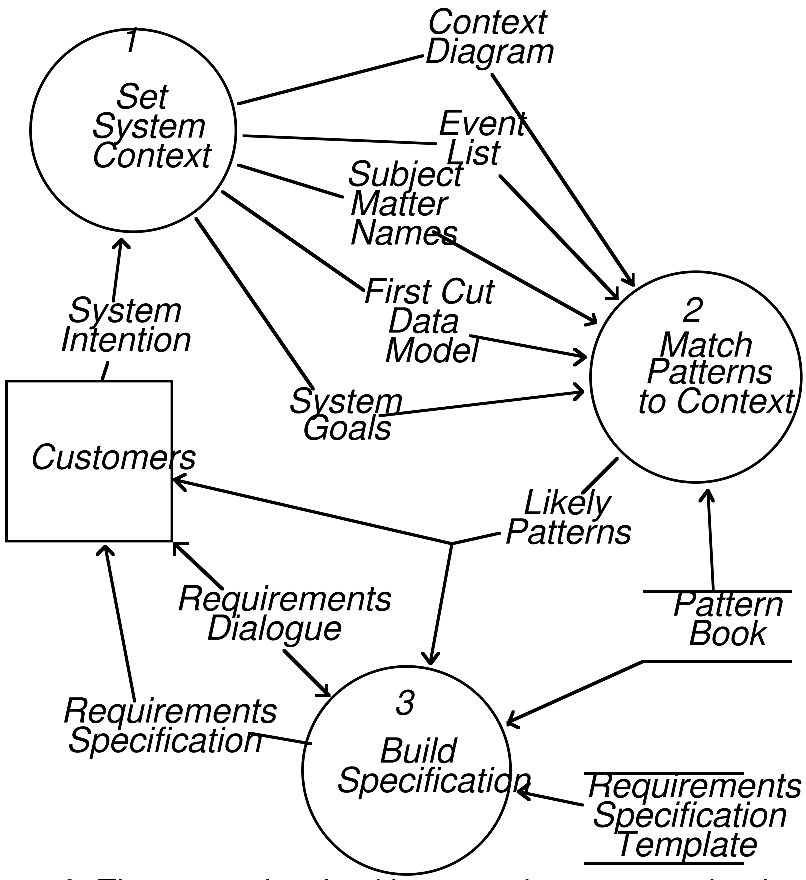


Figure 9: The stages involved in accessing patterns that have been captured by previous projects.

The stages you go through (defined in detail in Robertson 1994) are:

1. Set the context for your project. This defines the boundaries of your project and puts you on the road to identify relevant patterns.

2. Do an event/use case partitioning and list the event/use case names, input and output on the event list. This breaks your system into manageable, logically related chunks.

3. For each event/use case on the list look up relevant patterns by matching event/use case name with all or part of the pattern name. For instance, if you have identified an event called *Customer wants to buy groceries*, then you can match that with the pattern called *Customer wants to buy product*. Then your starting point is the detailed pattern for Customer wants to buy product rather than starting the analysis from scratch.

By using the event/use case as a pattern cataloguing tool and as a project partitioning tool you can progressively build up your pattern catalogue and access the appropriate patterns for your project. One benefit of this approach is the time you save by avoiding re-analysis and re-specification of processes that you have already analysed many times before. This is a considerable saving when you consider how long it takes to analyse and specify all the processing details for an individual event/use case.

Another benefit is earlier and more relevant feedback from users. You can use an event/use case process pattern as the starting point for requirements analysis. Tell the users that the pattern is a general statement of one of their business events. Walk the users through the model and ask them to tell you about the differences between their business policy and the pattern. The pattern acts as a prototype and helps to generate feedback that would normally take many sessions to discover.

Conclusions

Event/use case partitioning is widely used as an approach for partitioning large systems. By the process of abstraction we can build requirements process patterns for generic events/use cases. The resulting requirements process patterns can be categorised by event/use case name and by input and output definitions. Each requirements process pattern focuses on the processing for a single event/use case and the data that is necessary for that processing to take place. A requirements data pattern provides help in understanding a complex event/use case by focusing on the data independent of the processes. Context analysis and event/use case partitioning provides a way of identifying and using relevant patterns early in the life of the project.

References

Alexander, Christopher, Sara Ishikawa and Murray Silverstein. *A Pattern Language.* Oxford University Press, New York, 1977.

This fascinating book talks about how to make use of patterns when constructing buildings. By following the patterns, a builder will construct a livable and beautiful house. A similar approach to software construction would undoubtedly lead to more usable and maintainable systems.

Gause, Donald and Gerald Weinberg, *Exploring Requirements - Quality Before Design*, Dorset House, New York, 1989.

This doesn't teach any methodology. Instead it teaches the much more fundamental issue of why requirements analysis is important, and why it can never be completely automated.

Hayakawa S.I, *Language In Thought and Action,* George, Allen & Unwin Ltd., London, 1970.

Deals with the functions of language and with the relationship of language to thought and understanding. Treatment of abstraction can be readily applied to the idea of building requirements patterns.

Hay, David C, *Data Model Patterns, Conventions of Thought,* Dorset House, New York, 1995.

Structures common to many types of business are analysed in domains like accounting, material requirements planning, contracts and process manufacturing.

Jacobson, Ivar et al. Object*-Oriented Software Engineering - A Use Case Driven Approach.* Addison-Wesley, 1992.

Using the concept of the use case, this book makes process models accessible to object-oriented developers.

McMenamin, Steve, and John Palmer. *Essential Systems Analysis*. Englewood Cliffs, N.J.: Prentice-Hall, 1984.

This outstanding book introduces event partitioning and provides the groundwork for requirements process patterns.

Robertson, James and Suzanne, *Complete Systems Analysis: the Workbook, the Textbook, the Answers*, Dorset House, New York, 1994.

This book contains a complete project which you can work through as a way of learning analysis techniques. The book is partitioned so that it can be used either as a textbook, for management familiarisation or a step by step case study with explanations and worked examples.

Robertson, James and Suzanne, *Mastering The Requirements Process*, The Atlantic Systems Guild Ltd., London, 1996.

A seminar and workshop concerned with eliciting and specifying requirements.

Robertson, Suzanne and Kenneth Strunch, *Reusing The Products of Analysis,* Second international workshop on software reusability, Position paper, IEEE, Lucca, Italy March 24-26, 1993.

Contains a case study of how requirements patterns were built and used in the insurance industry.