Business vs. System Use Cases

Author: Martin Langlands and Charles Edwards
Version: 1.9 Date: 22 Feb 2009

Abstract

Use-cases are very widely used as the basic concept for specifying requirements of commercial information systems. However, one area that causes problems is distinguishing between “Business” and “System” Use Cases. The aim of this article is to shed some light on this issue by highlighting the *differences* between the two, and proposing a diagrammatic way of showing how they are *related*. This is illustrated using a more detailed and meaningful example than is used many introductory texts.

Introduction

Use-cases are now very widely used as the basic concept for specifying requirements of commercial information systems. However, there is still a huge amount of confusion in how best to apply the idea. One area that continues to cause particular difficulty is distinguishing so-called “Business” and “System” Use Cases.

Why has this confusion arisen? Well, we need look no further than the man who gave the Use-Case idea to the world: Ivar Jacobson. In his 1994 book “The Object Advantage” [JACOBSON94], he introduces the concept of “The Use Case” as: “A use case is our construct for a business process.” (p104) So far, so good. But four lines later, he says: “A use case is a sequence of transactions in a system …”. No wonder there’s confusion: does the concept apply to business processes or computerised systems?

A good example of this confusion came up recently when a client asked: "We are generating a Business Use Case Model for a project. The Project is mainly to develop a system that can enable users to be notified by WAP/SMS on their cell phone regarding their preferred stock prices, important Emails, news, weather etc. Now which element shows the ‘Cell Phone’ usage in the diagram? A business actor, a business worker, a business entity or a use case? Also, can Business entities be shown in Business use case diagrams?".

Now, when faced with such a question, we’re usually tempted to respond: “Read the standard literature!”. Two of the most popular books on Use-Cases, by Cockburn [COCKBURN01] and Bittner & Spence [BITTNER03], have a lot of useful advice on exactly this. However, it is still surprisingly easy to absorb all that these and other authors have to say, and still be lost when actually starting to work on a real project. The aim of this paper is to help those stuck in this bind.

We’ll come back to answer this client’s questions directly later in this article. But before doing so, we need to do a lot of preparatory work.

Two Concepts

OK, so if there’s a possibility of confusion, how should we apply the terms Business Use-Case and System Use-Case?

Well, first, it’s helpful to note that there is commonality between the two concepts: both define a pattern of repeatable interaction or behaviour that is intended to deliver a result of value to the Actor.

However, it’s equally important to note there *are* clearly two useful but distinct concepts here:

- A **Business Use-Case** is to do with "using a business": this recognises that businesses[1] are created and organised in order to *do things for people* – mainly customers, but also other "actors". So a Business Use-Case is a way in which a customer or some other interested party can *make use of the business* to get the result they want – whether it's to buy an item, to get a new driving licence, to pay an invoice, or whatever. An important point is that a single execution of a Business Use-Case should encompass *all* the activities necessary to do what the customer (or other actor) wants, and also any activities that the business needs to do before the process is complete from its point of view. (We'll see a little later an example of what we mean by this last point.) So the duration of a BUC execution can vary greatly, depending on its nature. Some BUCs, like withdrawing cash from an ATM, can be done in less than a minute; others, like ordering goods for delivery, or getting a new phone line installed, can take days, weeks or even longer.
- In contrast, a **System Use-Case** is a way in which a user of a *computer system* can *make use of the system* to get the result they want. This will typically be something we can readily imagine as being done in a single sitting on a single PC or other device such as an ATM or a mobile/cell phone, usually with a single UI, or a small number of closely-related screens such as a wizard, and taking maybe between a couple of minutes and a half-hour at most. Alistair Cockburn suggests that a useful guideline is the "coffee-break test": once the user has completed (a single execution of) the System Use-Case, s/he can take a coffee-break with a clear conscience. The system use case also avoids all manual issues such as "file the printout" or "phone the customer once the order is confirmed", etc.

A Realistic Example

Let's look at an example that allows us to focus on the differences. This example is a little more complex than many that are presented at the introductory level, but therein lies the root of much of the problem: examples used in many books and articles just have insufficient complexity to illustrate the kinds of thing that come up in real life, and often are too simple to allow us to see the differences between Business and System Use-Cases.

So let's take SupaStores, an imaginary grocery chain that allows customers to place orders on its website, www.supastores.com, and have the orders delivered to the customer's home. We'll look at an example Business Use-Case (BUC) and some System Use-Cases (SUCs) to highlight the key differences. We've assumed a basic familiarity with UML modelling concepts.

The Business Use-Case

What Business Use-Cases might be of interest for SupaStores? Let's focus on the most obvious:

- ***Buy groceries***, from the customer accessing SupaStores to completion and recording of delivery for that customer.

A first-cut Use-Case diagram for this BUC would look very simple (Fig 1). This shows the principal actor – the Customer – and one Use-Case. The result of value to the customer would be the correct groceries delivered to their door.

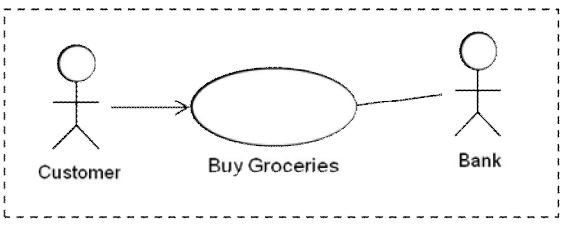

Figure 1: "Buy Groceries" Business Use-Case Diagram

We also need to show one supporting actor, the Bank, because it is a distinct business entity required to complete the

[1] Calling these "businesses" can itself be (unintentionally) misleading. A large part of the economies of all countries is conducted by organisations other than commercial businesses – local and central government agencies, charities and other non-profit organisations, etc. Their activities equally fall into this area. Strictly, we should be saying something like "Businesses and other organisations", or maybe just "Organisations"; but we'll stick with convention.

job: specifically, it needs to approve the customer's bank-card transaction. Note that we don't show on *this* diagram the "internal actors" in the SupaStores business that perform the use case – neither the employees such as warehouse and delivery staff, nor any tems (computerised or otherwise) involved – because they are "inside" the Use-Case. This Business Use Case Diagram describes "Black Box" behaviour; that is, it shows only the business interaction, not the internals of how the Business Use Case is implemented by SupaStores.

One of the most important things to understand about any given BUC is its *scope* – what triggers it, and what marks its completion. In this case, the event that starts this BUC is the customer accessing the SupaStores website to place an order. The event that completes the BUC is recording the details, such as the time and date, of the successful delivery. Note that the duration of this BUC could be from around a day to a week or more.

In order to see how BUCs and SUCs differ, and also how they relate to one another, we'll have to open up this BUC and take a look inside.

The first problem we meet is: UML, the modelling language that defines the Use-Case concept, says nothing about what the inside of a Business Use-Case (or any Use-Case, for that matter) should look like! However, a sensible approach – and one that is very useful, as we'll see – is to look at the sequence of steps, or *activities*, that constitute the BUC. Figure 2 illustrates this diagrammatically.

We've drawn this more-or-less as a UML Activity Diagram, with a very simple drawing palette:

- rounded rectangles for activities (aka steps)
- simple arrows showing activity sequence
- a bullet for the trigger start event, and a bullseye for the end event
- swimlanes representing which roles or organisational units are doing what activity – the Customer and the Bank, as actors external to the organisation, get their own swimlanes, as does each internal organisational unit involved.

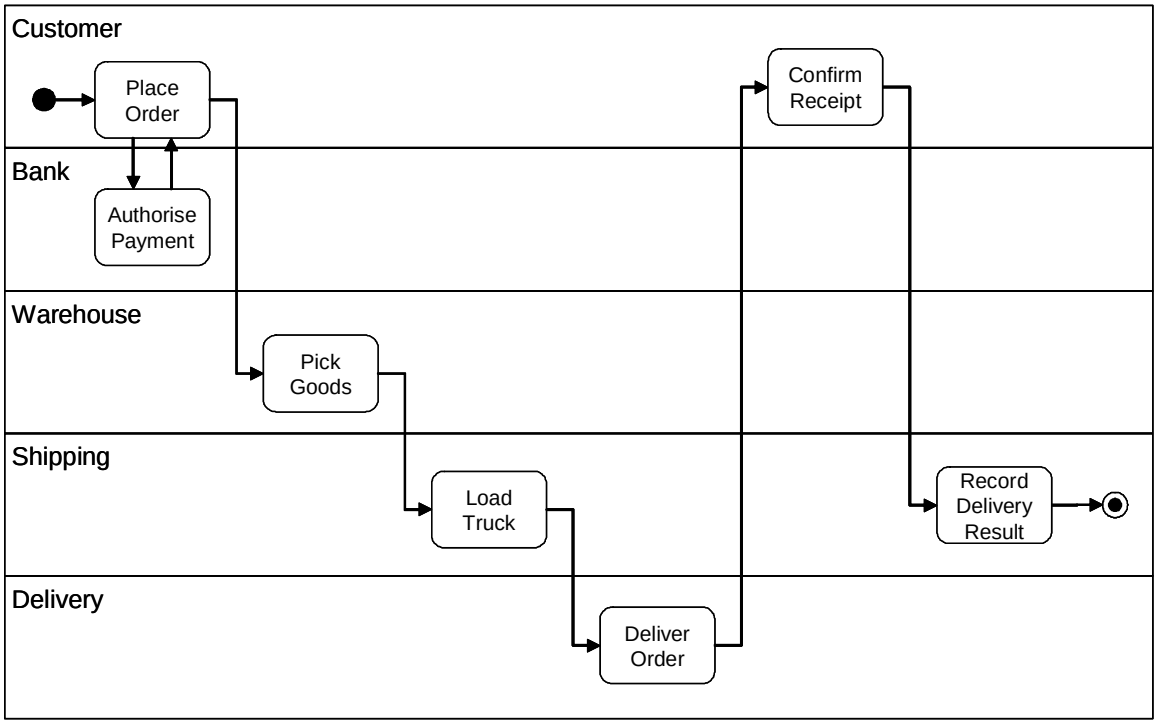

Figure 2: "Buy Groceries" Business Use-Case Activity Diagram

As mentioned, this is a non-trivial example, but one that we hope is readily understandable without having specialised domain knowledge. We've deliberately kept this as simple as possible, without getting into the complexities of different modelling approaches or notations such as Business Process Modelling Notation [BPMN]. Also not shown are the different possible alternate flows or terminations – eg. the BUC could terminate immediately after the first "Place Order" activity if the customer decides to cancel, of if the card payment fails; in "Deliver Order" the driver could be unable to find the address, or the customer could be out; and so on. Showing these would result in a considerably enriched, but more complex, diagram

Now, although the diagram sets out the basic steps and their sequence, we can clarify our understanding further with a textual description. Figure 3 shows the Use-Case description based on Cockburn's widely-used template.

Name : BUY GROCERIES Business Use-case

Brief Description **:** In this Use-Case, a Customer places a new order for delivery. Supa-Stores assembles the order, delivers it to the customer, and records the result of the delivery

Principal Actor : Customer

Precondition **:** Customer must reside in the country of the business to qualify for delivery.

Main Flow

Step Name	Description
Place Order	The Customer contacts SupaStores; selects the required quantities of products from the product catalogue; selects a delivery slot from those available; offers payment by an approved method; advises delivery address; and confirms all order details.
Authorise Payment	The Bank confirms that the offered payment is valid, and authorises the payment.
Pick Goods	For all confirmed orders for a forthcoming delivery run, the SupaStores warehouse finds the order items on the shelves; assembles the orders, noting any out-of-stock items or substitutions; packs them for delivery; and prints a Delivery Sheet for each order.
Load Truck	For a particular delivery run, the SupaStores Shipping department determines the planned delivery route, and loads the packed orders (accompanied by their Delivery Sheets) into the assigned truck in the correct order for the route.
Deliver Goods	A SupaStores Delivery driver makes a delivery run, following the planned route; for each order on the run, the driver finds and confirms the delivery address, and delivers the order to the recipient
Confirm Receipt	For each delivered order, the Customer confirms that the delivered order corresponds to the items on the Delivery Sheet (including any unavailable and substituted items), indicates the delivery time and order acceptance, and notes any comments.
Record Delivery Result	After the completion of a delivery run, the delivery driver returns the Delivery Sheets to SupaStores Shipping department, who record the details of each order, including the time of delivery, and any notes made by the customer.

Figure 3 : Text Description of Business Use-Case
"Buy Groceries"

This deliberately shows only what Cockburn calls the *Main Success Scenario* or *Main Flow* for this Use-Case (also often referred to as the *sunny day* scenario) – that is, it still omits any alternative or exception flows such as those leading to the different terminations considered above.

The textual description introduces some important subtleties that are not indicated on the activity diagram, such as the handling of out-of-stock items and substitutions. It also illustrates – in the step *Record Delivery Result* – an example of the point made earlier about "tidying-up" at the end of the BUC: although the customer has received the goods in the penultimate step, and the BUC is therefore complete as far as s/he is concerned, we can't regard it as finished from SupaStores' viewpoint until the final step is done.

Note that this is, in Cockburn's terminology, a "white-box" view of the BUC; we've gone beyond describing only the dialogue between the actors, to look behind the scenes and see how the BUC is implemented within the business. In this view, we've given a swimlane in the diagram to each Actor, and one to each SupaStores organisation unit involved.

A black-box view, showing only the interaction across the interface between the primary actor (the customer) and SupaStores, would be limited to the steps "Place Order", "Deliver Goods" and "Confirm Receipt", and on the diagram would need swimlanes (if we chose to show them) only for the Customer and SupaStores.

The intention at this stage is to show the activities that constitute the BUC – *what* gets done – without (yet) asking *how* they are done, in particular what computerised systems do or could support each activity.

The System Use-Cases

OK, now we've seen what a Business Use-Case looks like, let's take a look at the relevant System Use-Cases. This will allow us to see clearly the differences between the concepts, and how they are related.

First, let's make the (perhaps obvious) observation that we are certainly *not* going to have one SUC corresponding to this BUC. The scope and nature of one execution of the BUC is not something that we can conceive of being done in a single sitting in front of a computer. The customer may perceive it as being so, but the groceries won't arrive the moment they finish placing the order!

Secondly, in order to illustrate the SUCs here, we're going to have to be a bit more imaginative than with the BUC above. At a guess, Figures 2 and 3 would just about work for any home-delivery grocery store anywhere. However, the SUCs will depend on what systems SupaStores uses, and their functionality. So let's assume that they have a not-untypical mix of reasonably up-to-date systems, combined with some legacy applications, and with some level of integration between them. These will be discussed as we go.

Finally before looking at the SUCs themselves, note that in doing this we are now moving from the *what* gets done, to *how* it's done, in terms of the specific system support for each step. From our (invented) knowledge of SupaStores' systems, we can go through the business use case and identify the various system use cases that deliver this support.

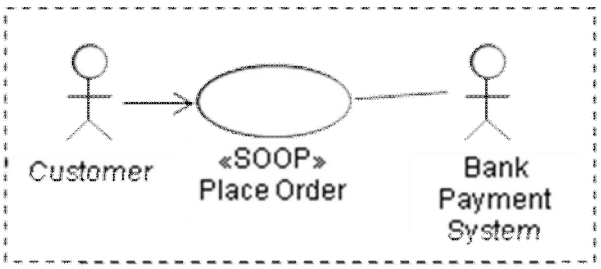

Figure 3: Use-Case Diagram for "Place Order" SUC

The first step – "Place Order" – is supported by one system use case, which has also been called *Place Order.* Figure 3, shows a simple Use-Case diagram for *Place Order*.

This system use case is implemented in the SupaStores Online Order Processing (SOOP) system, which delivers the order-entry pages to the SupaStores website, supported by an order database.

On the diagram, we've used a UML stereotype notation on the SUC to indicate the system, «SOOP», that implements the use-case. This system use case allows the customer – the primary actor – to select products into a virtual shopping trolley and choose a delivery time-slot.

Within this SUC, the customer also pays for the order using a payment card. This can't be done by SOOP alone; it needs to communicate with the banking payment system, which therefore needs to be shown as a separate, supporting, actor. (The primary actor initiates the system use case; supporting actors are called on by the system use case.)

This diagram looks a lot like Fig 1, which showed a Use-Case Diagram at the ***Business*** Use-Case level. However, note that the scope of Fig 3 is very different: it shows a ***System*** Use-Case that supports only the first step – "Place Order" – in the Business Process. Here, the system use case name is the same as that of the business use case step. This is reasonable because this system use case implements pretty much the entire business activity. Also, in the Business Use-Case diagram (Fig 1), we showed the *Bank* – a business entity – as a supporting actor, whereas we're now showing here the Bank *Payment System* as the corresponding supporting actor. This highlights the fact that business use cases are about *business* concepts, while system use cases are about *systems* concepts. "Well, obviously!" you say – but we suspect that a lot of the problems come from not applying this "obvious" principle.

We'll now go through each step in the business use case and ask: "What system use cases support this business activity?". Figure 5 shows the result.

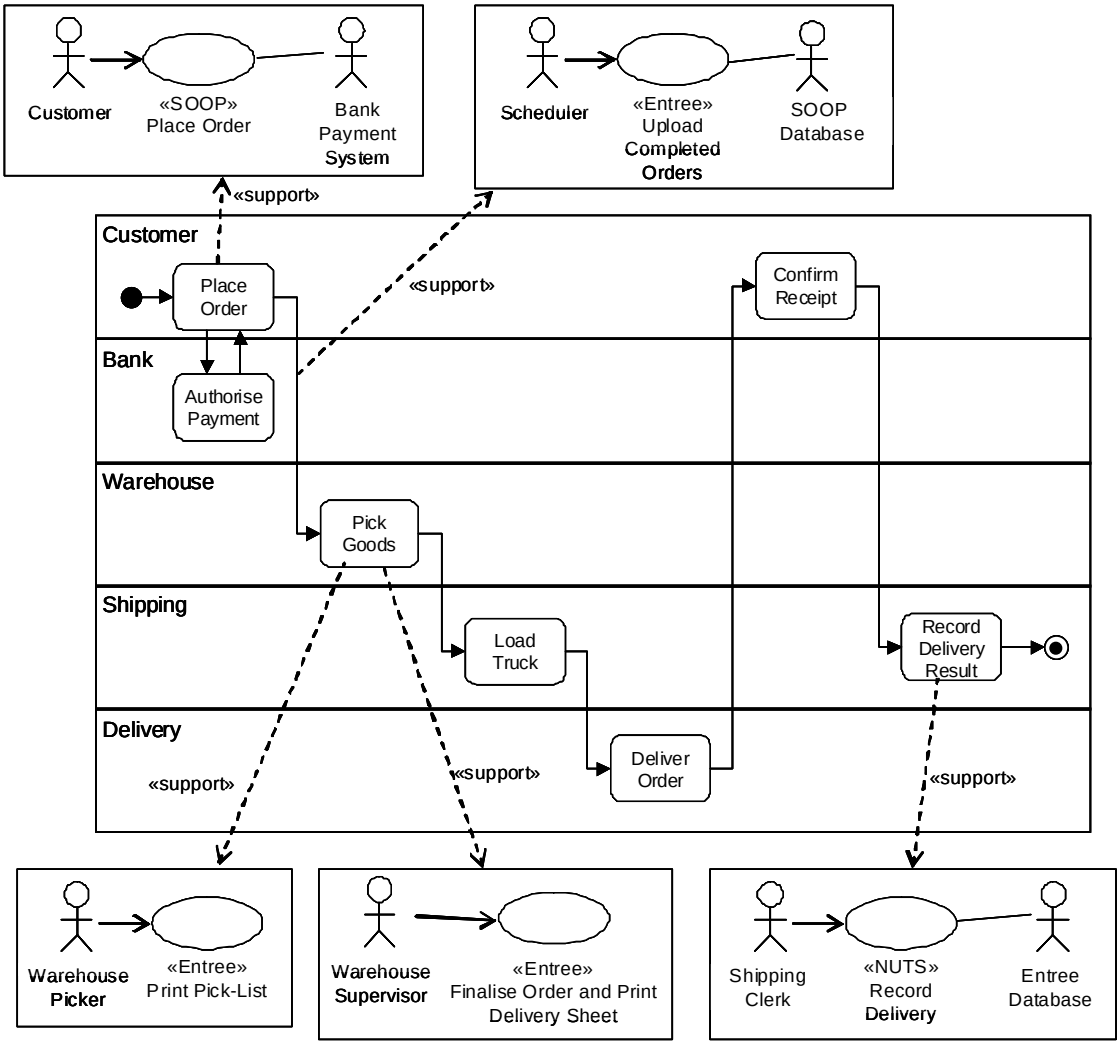

Figure 5: "Buy Groceries" Business Use-Case
Activity Diagram with supporting System Use-Cases

This is the same business use case Activity Diagram as Figure 2, but now with all the supporting System Use-Cases superimposed. The relationship between each system use case and the relevant elements of the business use case is shown by the associations labelled “supports”.

Note that this is a new kind of diagram, which we call a *Process / Use-Case Support Diagram,* or *PUCS Diagram.* We’ll come back to discuss this below. In the mean time, let’s continue with completing the picture.

The second activity on the BUC, “Pick Goods”, is performed by the SupaStores warehouse. This uses an older stock-management system based on a (hypothetical) package called “Entrée”. Among many other things, Entrée allows the warehouse staff to print off a hard-copy pick-list of orders due to be shipped in the next couple of hours. This is shown by the *Print Pick-List* SUC. After executing this SUC, the picker continues with the BUC step by going round the warehouse shelves, assembling the order, and marking-up the pick-list by hand as s/he goes to show out-of-stock items, substitutions and so on.

However, there’s something else that has to happen first: the order details need to get from the SOOP system to the warehouse’s Entrée system. This is done by a batch job in Entrée, scheduled to run twice daily, that queries the SOOP database. The batch job is represented by the SUC *Upload Completed Orders*. This has two actors, both systems, rather than human actors: the primary actor is the Scheduler, and the SOOP Database is a supporting actor. Finally, note that the “supports” association from this SUC is not to an *activity* on the BUC; rather the SUC supports the *transition* between the Place Order and Pick Goods activities.

Now, going back to where we were at the “Pick Goods” activity: once the warehouse employee has finished picking goods for each order, s/he hands back the marked-up pick-list to the warehouse supervisor who uses a further Entrée System Use-Case, *Finalise Order and Print Delivery Sheet*. This allows the supervisor to flag in the system any items that couldn’t be found on the shelves, indicate any substitutions, and print a delivery-sheet for the truck driver. So there are two SUCs that support the same BUC Activity. This might indicate that the activity needs to be split into two on the diagram (this iterative modelling between development of BUC and SUC models is typical), but we’ll leave it as it is for now.

The next three BUC activities – “Load Truck”, “Deliver Order”, and “Confirm Receipt” – have no systems support. Route planning is done by the delivery staff and driver using their local knowledge and a street-map. The delivery-sheet printed off by the warehouse supervisor provides all the information the driver needs en route, but reading and signing the paper can hardly be counted as a “system uses-case”.

The final activity in the BUC – “Record Delivery Result” – does have system support. The clerk in the shipping depot records post-delivery details from the sheet into a third system, used to track completed orders and also to record utilisation of delivery vehicles, called the National Utilisation and Tracking System (NUTS). This accesses delivery data from the Entrée database, which is therefore again shown as a supporting actor.

Note that the SUCs we’ve listed above as supporting the BUC steps are far from being the only set we could imagine. In fact, we’ll look later at what a different set could look like, and how this would affect the overall picture.

Inside the System Use-Case

Earlier, we opened up our example Business Use-Case to see what one looks like inside. What happens when we do the same thing with a System Use-Case?

We’ve taken the “Place Order” SUC as the example to illustrate. Figure 6 shows the first-cut SUC Description. Again, as for the BUC before, we’ve shown only the Main Success Scenario or Main Flow.

Note that we could have drawn a UML-style Activity Diagram similar to Figure 2 to represent this SUC. However, at this stage, it would not be very interesting: just a single se-

quence of activities, more-or-less alternating between Actor and System swimlanes had we chosen to show them. Drawing such a diagram would, however, get much more interesting and valuable once we include alternative and exception flows.

Name : PLACE ORDER System Use-case

Brief Description **:** In this Use-Case, the Actor uses the SOOP system to place a new order for delivery. The actor selects the required products, chooses a delivery slot from those available, and makes a card payment for the order.

Principal Actor **:** On-line Customer

Precondition **:** The Actor has successfully logged on to SOOP as a valid and registered customer

Main Flow

Step Name	Description
Trigger	The Customer chooses to place a new order
Select Products	The Customer selects as many products as wanted into the "shopping-basket", choosing from those offered, and entering the required quantity ordered of each.
Indicate selection complete	The Customer indicates that the product selection is complete.
Display total price	The System calculates the indicative order price, applying multi-buy discounts, and displays the resulting total
Accept price	The Customer indicates that the price is accepted.
Display available delivery slots	The System finds and displays available delivery-slots for scheduled delivery runs covering the address held for the customer
Choose delivery slot	The Customer selects the desired delivery-slot from those presented.
Present stored payment card details	The System displays details of payment cards held for the customer and requests the customer to choose a card
Confirm payment card	The Customer selects one of the presented cards
Validate card payment	The System contacts the Bank Payment System, which validates the card payment
Present order details	The System presents full details of the order (shopping-basket with indicative prices; delivery address; selected delivery slot; card payment details) for the customer to print if required
Email order details	The System sends an email of order details to the email address held for the Customer
End	The Use-case ends

Figure 6 : Text Description of System Use-Case
"Place Order"

Superficially, Figure 6 is very similar to the Business Use-Case description in Figure 3. Both show an interaction, or dialogue, between an actor and the conceptual system that implements the Use-Case.

However – and this is critical to the central point of this article – the fundamental nature of the two Use-Cases is very different.

- In the BUC example, this conceptual system that implements the Use-Case" is the SupaStores *business* itself, whereas in the SUC it is the SOOP *system*.
- The nature of the interactions is completely different, too: in the BUC, they are "real-world" interactions involving physical people, goods, trucks and so on, while in the SUC they are probably[2] conducted via a PC screen, mouse and keyboard.
- Clearly, the duration of the BUC and SUC are very different – a few days in the first case, a few minutes in the second.

Note that, in contrast with the BUC in Figure 3, Figure 6 is a *black-box* view of the System Use-Case: it shows only the dialogue between the actor(s) and the system, with no internal implementation details. But this does not affect the fundamental points of difference.

The Key Differences

We're ready to summarise the key differences between the two concepts.

Aspect	Business Use-Case	System Use-Case
Who's the Primary actor?	Mainly a **business actor** e.g. customer; maybe other external party (regulator, shareholder) or an internal party (manager, etc)	Mainly a human user who initiates system behaviour; maybe another system, "scheduler" etc. But by definition a **system actor**
What's the use case for?	Something the actor wants to get done by using the **business / organisation**	Something the actor wants to get done by using the **system / application**
Who / what else may be involved?	May involve interaction with other **external business parties** as supporting actors	May involve interaction with **other systems** internal or external to the organisation.
What does it describe?	Describes an **interaction** involving the primary **actor**, the relevant parts of the **business**, and any supporting actor(s), in terms of their **business behaviour**	Describes an **interaction** involving the primary actor, the relevant parts of the **system**, and any supporting actor(s), in terms of their **system behaviour** . In the case of the primary actor, this means only their actions **detectable by the system,** such as making selections, supplying data etc.
How's it executed?	May involve **many organisation units**, systems (or not), technologies, manual / mental procedures etc.	Executed by **automated steps** in the system
Duration	Of **Varying duration** - May be very brief or very long-duration.	Typically quite **short duration** (Cockburn's "coffee-break" rule)

When faced with a particular modelling task I a real-life project, these differences helps us to decide: "Which one am I modelling here?".

Make sure you know the answer, and stick to it. Get yourself into the appropriate mindset, and don't mix the business and system views in the same Use-Case. Once you've got

[2] We could imagine this done another way – eg. with an interactive TV and a remote control, or even, at a stretch, via a voice- and keypad-activated phone system. The key point is that the actor is interacting with a *computerised system*.

them sorted out, you can show how they're related using a Process / Use-case Support (PUCS) Diagram.

Requirements or Design?

We said above, in starting to discuss the System Use-Cases that support our Business Use-Case, that "we are now moving from the *what* gets done, to *how* it's done". This raises an issue that often causes problems, and in doing so typically generates more heat than light. The question that's asked is: "If we're now moving to the *how*, are we not moving from *requirements* to *design*? And if so, isn't there a conflict in that Use-Case Descriptions are supposed to be a *requirements* artefact?".

This topic has been discussed extensively (try Googling for "requirements vs design"), and we don't want to reiterate all the arguments here. But to summarise our view: Use-Case Descriptions are *both* Requirements *and* Design artefacts.

How can this be?

Bear in mind that there's a progression from coarse-grained outline business requirements through to implementation of a software solution. (And no, we're not advocating a waterfall approach here: this progression can happen within any fine-grained "unit of requirement", such as a User Story; and it's OK to iterate round this progression.) At each stage, we take stated requirements and apply a range of factors – logical considerations, technical guidelines, best practices, constraints and so on – to come up with a design at the appropriate level of detail. That design becomes the requirements statement for the next level of design.

This "requirements vs design" issue is closely related to the viewpoints of the various stakeholders in a project. To a software developer, the Use-Case Description in Figure 6 might look very much like a ***requirements*** statement, and a fairly un-detailed one at that. To her, it's just waiting to be tackled with some decent design, covering UI layout and behaviour, object / component design, system integration, performance, security and so on. But to the business manager responsible for the new on-line order service – whose main concerns may be doing a deal with the right payments operator, or ensuring that warehouse and delivery people understand the impact of the new service – Figure 6 looks like pretty detailed system ***design***.

And the thing is – they're both right!

The Process Use Case Support (PUCS) Diagram

In order to fully appreciate the value of the separate Business and Systems Use-Case concepts, we need to understand not only where they *differ*, but also how they are *related.* The fundamental nature of the association is:

System Use-Cases *support* Business Use-Cases

– meaning that the System Use-Case provides some automated means of doing part or all of the Business Use-Case.

We are proposing here a new diagram-type, which we call the *Process / Use-Case Support Diagram*, or PUCS Diagram, to provide an immediate visual representation of this support. Figure 5 shows an example. It is a simple but effective addition to the UML diagram set, using existing UML concepts, and with additional stereotypes that could easily be implemented in a simple UML profile.

Each PUCS Diagram is drawn for one Business Use-Case. The key elements of the diagram are:

- A UML Activity Diagram showing the steps (activities) in the Business Use-Case (BUC) and the transitions between steps
- A UML Use-Case Diagram for each "supporting" System Use-Case (SUC)

- An association, named “supports”, from each SUC to each business activity / transition that it supports. (Some activities or transitions may have no such SUC support.)

The nature and extent of the support will vary from business to business and from one system to another. For example, the support given by a SUC to a business activity can vary

- … from simple data presentation and / or capture, such as a straightforward data-entry UI with limited validation …
- … through partial decision-support, such as an insurance rating tool …
- … to full implementation of the whole step, such as an automated share-trading system.

These variations could be indicated on the PUCS diagram with stereotyping of the “supports” association, although care needs to be taken not to attempt to show too much on the diagram.

What were we doing in the above example?

In following the SupaStores “Buy Groceries” example, the astute reader may have asked: “Why are we doing this? What’s the ‘project’ here?”.

This is a fair point. From what we said, many of the SUCs in question are realised by old legacy systems. When those systems were first developed, there’s a very good chance that Use-Case Analysis did not figure as a technique. So why are we going back in time now to represent their functionality as SUCs?

Well … apart from the obvious answer “to illustrate this article”, a real-world example would be a project that wanted to improve system support for a part of the process – say, the back-end “delivery” steps. Let’s say that the SupaStores executive in charge of on-line orders has identified a problem that the keying of the delivery-sheet details into the NUTS system (in the step “Record Delivery Details”) is error-prone and often done late, resulting in poor management information, and incurs extra staff costs to do the data entry. She has sponsored a project for IT to look at the problem and come up with a solution.

If we look at the problem in the fairly narrow terms stated by the project sponsor, we are likely to come up with a narrow solution. If we don’t have the benefit of the BUC model, and just focus on the use of the NUTS system by the shipping clerk, we might conclude that the problem lies partly in a poor design of the delivery-sheet, or an unfriendly old character-based NUTS UI, and tackle these issues. However, this would be missing an opportunity.

We can propose a more imaginative solution by taking a wider view to look at the whole Business Use-Case as in the example. Now we can see where the recording of the delivery-sheet fits in the bigger picture, and ask: “What opportunities are there to improve the whole process, by enhancing systems support, and / or by other changes?”.

We could, for instance, propose giving the delivery-drivers web-enabled hand-held devices which access and display delivery lists, and which are used to capture customer signatures. This would be supported by a new system, SWEET (SupaStores Web-Enabled Enhanced Tracking), that would maintain a database of current deliveries. Now we can completely eliminate the costly and error-prone “Record Delivery Details” step. SWEET realises a new “Confirm Receipt” SUC, in which the customer (the SUC actor) uses the handheld screen to record delivery details, which are immediately recorded on the SWEET database. Figure 7 shows the result on a new PUCS Diagram, showing the changed BUC Diagram and a new set of supporting SUCs.

We can also see opportunities for further improvements – for instance:

- We can help the delivery driver view the deliveries in a delivery run, perhaps displaying these dynamically on a local-are map on the handheld (SUC “Display Deliveries”)

- If we further add GPS functionality to the handheld, allowing SWEET to track the location of each delivery truck, we can provide the customer with a "when's my delivery coming?" function that she can access through her own home PC (this supports a different BUC, so it's not shown on Fig 7).

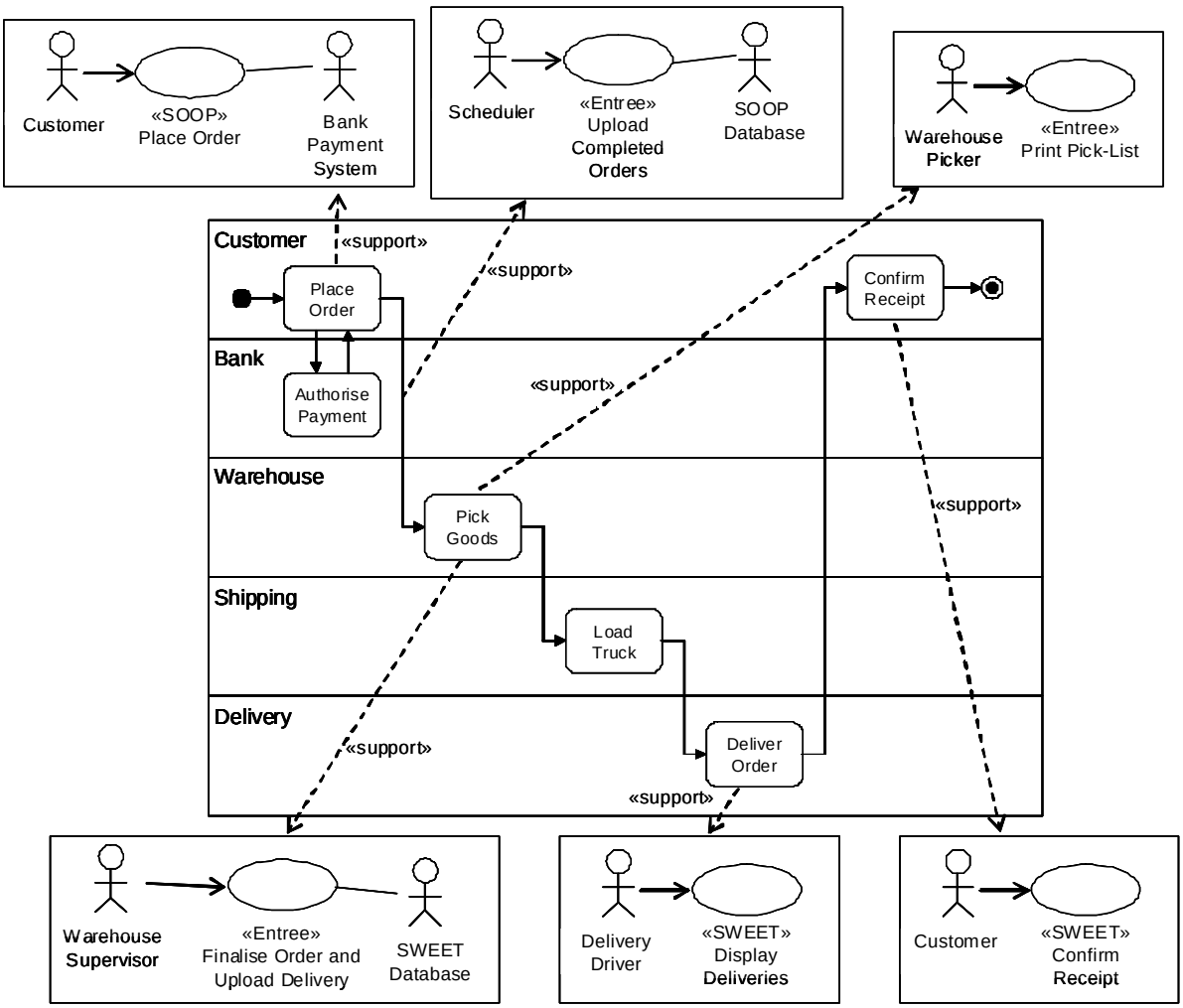

Figure 7: "Buy Groceries" Business Use-Case
Activity Diagram with supporting System Use-Cases – Proposed enhancement

Of course, all of these ideas, and any others we come up with, need to be subject to a proper business assessment, including a cost / benefit analysis, and not all will necessarily be approved.

But what we have done is to show a straightforward but powerful representation of both the underlying *business process* and the current and potential *systems support* for that process. This representation is readily understood by business and systems people, and is an ideal tool for discussing, scoping and planning the systems-development aspects of a business change project. Newly-defined system use-cases identified this way can, of course, go on throughout the project as the basis of detailed specification, design and implementation. In addition, the fact that our legacy systems may not have been originally specified or implemented using explicit SUC modelling does not detract from the usefulness of the System Use-Case concept for describing the functionality of those systems retrospectively.

Three Caveats

We have attempted in this article to help modellers with the sometimes tricky problem of distinguishing Business Use-Cases from System Use-Cases. However, there are still some issues that can lead to possible confusion.

"The Business as a System"

Sometimes we encounter the view that "there's no real difference between the two – after all, the business can be viewed as a system, can't it?". This is as much a question of definition of the terms: yes, we can say this, if by "system" we mean something like "a collection of interacting elements, organised as a whole and collaborating to achieve a purpose". While defensible – and there is, of course, a whole range of disciplines related to understanding "systems" with this meaning – it seems to us unnecessarily confusing in this context. When we say "system" here, we specifically mean *computer(ised)* or *automated* system, and not *"business-as-a-system"*, and we urge readers to do the same.

"System Boundary = Business Boundary"

In many cases, the scope of the System Use-Case is exactly the same as the Business Use-Case, and the SUC completely implements the BUC. Another way of saying this is that the boundary of the system across which the SUC interaction takes place is the same as the BUC boundary – they are "coterminous". Examples include:

- Web sales of downloadable software, music or other digital material
- On-line flight check-in
- Withdrawing cash from an ATM (but not all possible ATM-based BUCs – for example, "requesting a cheque-book")
- Making a purchase from a vending machine
- Connecting a phone-call in an automated exchange

The key point about these BUCs that makes them completely implementable in a coterminous SUC is that there's no need for separate steps to make decisions or handle physical goods, or any inherent delays between steps. The SUC may need to involve other systems, as further secondary actors – for instance, to actually retrieve and deliver the software or music, check and update account balances and so on; but it's still the case that one System Use-Case implements the complete Business Use-Case.

Examples such as these are, of course, common and important, and are likely to become more so as digitisation of products, and the world in general, gets wider. They also tend to be beloved as examples by writers of text books and articles. Unfortunately, this does often obscure the real distinction between BUCs and SUCs – because in these cases there is often little value in writing descriptions of both.

However, there are of course still very many cases, such as the SupaStores example, where the SUC is not going to completely implement the BUC (short of a Star Trek transporter device to deliver cereals, steaks and kitchen-roll digitally).

"One Level Too Low"

A third problem that we sometimes encounter in distinguishing BUCs and SUCs is that modellers misapply the terms. This tends to be more common with technically-oriented people – perhaps developers doing Use-Case modelling – rather than "business analysts".

We've observed that they apply the term "Business Use-Case" to what we call a "System Use-Case"; so, for example, they'd define "Place Order", "Print Pick-List" and "Record Delivery" as *Business* Use-Cases. The terms have been moved "one level down" as shown in Fig 8.

We say ...	***They say ...***
Business Use-Case	???
System Use-Case	Business Use-Case
Tech Design / implementation Artefact	System Use-Case

Figure 8 : Terms moved one level down

The thinking seems to be that "these are things that the business user touches", so they must be "business"-things. In this view, the term "*System* Use-Case"

is then applied to things that happen *inside* the system – in effect, the things that (in our view) would appear once we opened up the description of the SUC to the white-box level. They correspond to technical design or implementation artefacts such as components, operations, or APIs that have no direct element of actor interaction or dialogue, and therefore can't count as Use-Cases at all. (We might label these *mis*-use cases.)

A problem with this misuse of the terms is, of course, that there's then no term left for what we have called a Business Use-Case. Unfortunately, since this arises from a technically-focused mindset, this isn't seen as presenting a problem – the true "business" thinking is way out of sight in any case. What's needed is to take a step back and appreciate that the system solution is there to support, and is intimately related to, a view of what's going on at a true business level, before we consider systems issues at all.

Revisiting the Original Question

We're finally in a position to discuss our client's question that we raised at the start of this article: "We are generating a Business Use Case Model for a project. The Project is mainly to develop a system which can enable users to be notified by WAP/SMS on their cell phone regarding their preferred stock prices, important Emails, news, weather etc. Now which element shows the 'Cell Phone' usage in the diagram? A business actor, a business worker, a business entity or a use case? Also, can Business entities be shown in Business use case diagrams?"

Unfortunately, there's no single answer to this. What we need is a longer discussion with our client. It seems possible that the questioner is suffering from the "one level too low" problem, among others. However, making some guesses, the response would be something like the following.

- Firstly, an important point is that the project team seems to be placing too much reliance on Use-Case Diagramming as a technique. Use-Case diagrams as such show very little; don't get too hung up on what you can expect to get out of them, and don't think that by drawing Use-Case Diagrams you're "doing Use-Case modelling". Move on to textual descriptions as soon as you can, initially at the "use-case brief" level and then on to main scenarios and finally alternative flows.
- To take the last part of the question next – "*can Business entities be shown in Business use case diagrams?*". The answer is most emphatically "Yes!". Business entities, such as the Customer and the Bank in our SupaStores example (see Figure 1), are critical elements of a BUC Diagram. Just don't start to think of "cell phones" as business entities!
- The key question here is: What *is* the core Business Use-Case? One possibility is something like "Advise Customer of Notifiable Event", where the primary actor is internal to "us" (ie. the company offering the service) and the customer – and probably the phone network provider – are supporting actors.
- Having decided this, create a white-box Business Use-Case Description – in both diagrammatic and textual forms; then determine what System Use-Cases are needed to support each step. The relationship between the BUC and SUC should be shown with a PUCS Diagram.
- Finally – and to answer the core question as posed: "*which element shows the 'Cell Phone' usage in the diagram?*" – we'd say: None of them! The Cell Phone should not appear at all on any of these diagrams – not the Use-Case diagram, nor Activity diagrams, nor PUCS diagrams, at either Business- or System Use-Case levels. The Cell Phone itself is simply the hardware that supports parts of the overall picture. To show it would be as inappropriate as showing the user's PC on the "Place Order" SUC Diagram, or indeed the "Warehouse Building" or "Delivery Truck" on the BUC Diagram in the SupaStores example.

References

[BITTNER03] Kurt Bittner & Ian Spence, “Use case Modelling”, 2003, Addison-Wesley, ISBN 0-201-70913-9

[COCKBURN01] Alistair Cockburn, “Writing Effective Use Cases”, 2001, Addison-Wesley, ISBN 0-201-70225-8

[JACOBSON94] Ivar Jacobson et al, “The Object Advantage”, 1994, Addison-Wesley, ISBN 0-201-42289-1

[BPMN] Object Management Group – see http://www.omg.org/bpmn/

Contact details

Martin.Langlands@blueyonder.co.uk

Charles.Edwards@processwave.com

www.ProcessWave.com and www.AgileEA.com

