RM13: Practical application of use cases to a real-time system

David Hanslip

July 07, 2004

Derived from a RUC 2003 presentation, this article presents a real-world example and discusses problems encountered and lessons learned during the application of use cases to the specification of a real-time system.

Background

This article derives from the following session of the 2003 Rational User Conference in Orlando, Florida.

RM13 - Practical Application of Use Cases to a Real-Time System Wednesday, Aug 27, 2:45 - 3:45 David Hanslip, Software Engineering Specialist, Rational Software, IBM Software Group

Overview

This article is based on work that I've done over the last year with a client developing a real-time control system.

The objectives of this article are, first, to highlight the relevance of use cases to specifying real-time systems, and second, to illustrate how we can develop a use-case model that delivers the benefits we would normally expect from using use cases.

The first thing I want to look at is why we need an article such as this, and then we'll look at what's special in terms of using use cases to describe real-time systems.

After establishing the need for this article, I'll describe the project and its use-case model. I'll highlight some of the interesting and significant characteristics of the use-case model, and look at how it benefited the project. Finally, I'll discuss some lessons learned and present some recommendations.

Why do we need this article?

There's a perception that use cases are only useful when you are describing a highly interactive system, typically an IT system, such as a bank system. In a bank, you might be processing a mortgage application, for example, and there is a lot of user interaction. With a real-time system, there might not be a lot of user interaction at all. However, if a real-time system has significant
externally visible behavior, then use cases are still useful in specifying it, even if, in some of the use cases, the user selects some settings and then tells the system to perform some function, after which there's very little further interaction.

While working with customers over the years, I've found that use cases are widely misunderstood. This is despite the many books, papers and training courses available on the subject. Perhaps, the many books, papers and training courses actually contribute to the problem, since they provide considerable variation in approach. The example that's often used in the literature is the ubiquitous Automatic Teller Machine. Such an example may be useful to illustrate a point, but is often not particularly helpful when dealing with a large and complex system. How do you deal with the kind of problems that arise when working with a real-time system? It really helps to have some real world examples, and this article will provide one.

What's special about real-time systems?

A real-time system is one in which timing is critical. If a system doesn't meet its timing requirements, it could be life threatening, as in the loss of an airplane for example. In the system I'll describe shortly, failure to meet timing requirements could result in damage causing loss of production and losses totaling millions of dollars.

Real-time systems can have minimal interactions with the user, and, again, this is a consideration where specifying such a system with use cases is concerned. Real-time systems can be highly algorithmic, and use cases may not be the best way of documenting algorithms. Typically, a use case will refer to an algorithm that is documented elsewhere, leaving the use case specification rather empty.

If your system has significant externally-visible behavior however, then use cases are a great vehicle for documenting the requirements. The system I'll describe here has a lot of externally-visible behavior. All the same rules that apply to documenting IT systems using use cases apply here.

The product transport (PT) system project

The client is one of the world's largest iron ore producers with an iron ore export facility in the Northwest of Western Australia at Port Headland. Iron ore arrives on railroad cars from which the ore is transported by a network of conveyors, positioning devices, and weighing devices to stockpiles. Later, the ore is picked up from stockpiles and again transported by conveyor to ships.

Everything about this system is big. Some of the conveyors are 7 kilometers long and can take up to 15 minutes to start. A complex control system is required to run the conveyor network. One of the requirements on the control system is a reduction in the number of people required to run the system. Another is to reduce the level of training required of the people who run the system. Figure 1 is an aerial photograph of the facility.
Figure 1. Aerial View of the Port Hedland facility

There is a ship's berth at the top left corner. You can see the railroad lines coming in from the right. The car-dumping facilities are in the near-vertical line in the middle. There is a crusher up towards the top of this line. You can see the stockpiles in the North and South yards, conveyors running between the car-dumpers and crusher, and crusher and stockpiles. You can see the reclaimers which pick up the ore from the stockpiles and take it to the ships. The reclaimer buckets would carry a large car. The conveyors are capable of moving 10,000 tons of ore an hour in this system!

There are a number of requirements that significantly increase the complexity of this system and make it very interesting. The control system has to maximize productivity while avoiding spillage. Export capacity is to increase from 67 million tones per annum (mtpa) to 81mtpa by 2004 with scope for expansion to over 90mtpa by 2011.

Fundamental concept: The route

A route consists of a set of conveyors, positioning devices, and feed devices selected and coordinated to move ore from a source to a destination. Routes can be independent or share equipment to allow splitting to send ore from a single source to multiple destinations, or combining, taking ore from multiple sources to a single destination. Operators create, maintain, and delete these routes. Once an operator requests that a route be started, the system takes over and starts everything in the right order, dealing with conditions like there being incompatible product already on the conveyors.

In order to meet the productivity requirements, the control system adopts aggressive starting and fault-handling strategies. Whereas the existing system starts one conveyor in a route at a time, and when it is up to speed, starts the next one and so on, the new control system can start all the conveyors in a route simultaneously - keep in mind that we have conveyors with different lengths, and therefore start times, and capacities. When a fault occurs, the new system attempts to shut down the minimum amount of equipment consistent with avoiding spillage, because it takes up to 15 minutes to start a long conveyor. Wherever possible, equipment is left running until there
is a possibility of ore spilling. The system needs to synchronize Routes to send ore to multiple destinations and combine ore from multiple sources to generate blends.

Handling equipment faults is fully automated. When a conveyor is transporting up to 10,000 tons of ore an hour, you can imagine the results if there were a fault and upstream conveyors didn't stop quickly enough. The result is not something you could clean up with a wheelbarrow! You would have to bring in some heavy equipment to clean up the spill, and it takes a long time to do that. It would be even worse if for example, a fault caused ore to pour onto the deck of a ship.

**The PT system use-case model**

This work is based on both the IBM(r) Rational Unified Process(r) (RUP(r)), and the work that's described in the book, "Use Case Modeling", by Kurt Bittner and Ian Spence. Use cases are an alternative way of writing requirements. In the past, we have written declarative statements based around the word "shall." I worked on a defense project a few years ago where the contract included 22 volumes of requirements, and nobody could really understand them all. It was very difficult to make sure they were all correct and consistent. Use cases help us address those sorts of problems. Use cases put the requirements in the context of the value the system provides to the user and/or stakeholders. The individual use-case statements are presented in chronological order eliminating all the contextual baggage that traditional "shall"-based requirements have to include.

Only the salient features of the use-case model are presented here because the complete model is large and complex. The use-case specifications are long and very detailed.

**Identification of actors**

It's very important to focus on identifying the actors at the start of this whole process because that bounds your system, clearly indicating what is in-scope and what is out-of-scope. We don't want to be developing things we're not being paid for! In the PT System, the Reclaimer is a system that we interact with. While we can direct the Reclaimer what to do, we have no control over how it does it. We can't modify that. Since we have no control over the Reclaimer, it is outside our system boundary, and modeled as an actor. We have Ship-Hatch Loaders, Stackers, Sampling Stations and Dust Control Systems - all are external systems and are modeled as actors. Figure 2 shows a partial diagram of the actors for this particular system.
Figure 2. PT System Actors

In Figure 2, the actors are grouped using UML packages. On the left is a package of Operators. The most important one is at the top left. The CRO is the Control Room Operator. The other one of interest on bottom left is the RSMT (the Root System Maintenance Technician).

There is another package containing all the different equipment actors. Some of the use cases talk generically about equipment that we manage, and then there are other use cases that refer specifically to different types of equipment.

At the bottom of the figure, there are some External Systems with which we interact. The one on the left is the PMAC, which is an existing system which hosts our user interface. If it were just a user interface as far as our system is concerned, we wouldn't need to document it as an actor because the actor is actually the CRO. But it is an actor, as far as we are concerned, because we have to send information to it and collect information from it as well.

The Power Load Management System (PLMS) is interesting. The port facility at Port Headland has its own power station! When you are starting a conveyor that is 7km long, and may have thousands of tons of ore on it, it requires a lot of electricity. Hence, the starting of conveyors is actually staggered in order to spread the load on the power station. Whenever you want to start a conveyor, the PT System has to ask permission from the power station.

Identification of use cases

Use cases are simply an alternative means of specifying requirements. Firstly, let's revisit the definition of a use case:

A Use Case defines a sequence of actions performed by a system, that yields an observable result of value to an Actor or Stakeholder.

The "observable result of value" is important to keep in mind when identifying the use cases of an IT system, and it is here too. We are looking at what value this system provides to the actor or stakeholder. In our example, we have the controller room operator who is the actor, but the
company is actually getting the value by virtue of the system moving product from one place to another. The "sequence of actions performed by the system" are statements in a use case that describe what the system does. Each of these is a requirement on our system. Finally, a use case is a complete and meaningful flow of events. Combining this notion of "complete and meaningful flow of events" with "an observable result of value" helps guard against identifying snippets of functionality and calling these use cases.

Use cases and actors are represented graphically in a use case diagram. Figure 3 shows the top-level use-case diagram for the PT System. You can quickly see that PT use cases fall into 4 groups: Operation, Administration, Configuration and System Startup.

**Figure 3. Top-Level Use-Case Diagram**

Use cases that relate to operation of the PT System are shown in Figure 4.

**Figure 4. PT System Operational Use Cases**

The primary use case for the PT system is "Transport Product." Just from the name of this use case, you can see that the value that this system provides to its users and stakeholders is the ability to move product from one place to another. This software could be used in other applications, not just in transporting iron ore. It could be transporting tablets in a pharmaceutical factory, for example. Transport Product will be a big use case, but this captures the essence, the reason for the existence of our system, the value that it provides for our actor and our stakeholders.
If Figure 4 we see the interaction with the Power Load Management System (PLMS), requesting permission to start equipment and we see that the use case interacts with equipment that we are controlling. We could show each equipment actor but the diagram would become very cluttered.

In the Figure 4, the arrow from the CRO to the Transport Product use case, indicates that this operator initiates or starts this use case. The arrows from the use case to the equipment and PLMS actors indicate that the use case, or the system, talks to the equipment and talks to the power load management system. The arrow from the Ship Hatch Loading System (SHLS) indicates that SHLS initiates interaction with the system - specifically, the SHLS requests gaps in the ore flow to allow the loader to move to another hatch.

Use cases relating to administration, configuration, and startup of the system are shown in Figure 5.

**Figure 5. Use Cases relating to Administration, Configuration and Startup**

One of the things that we have to be able to do with this system is upgrade the software with minimal interference with the operation of the system. Therefore, we have a "Perform Software Upgrade" use case. We also have a "Start System" use case - system startup use cases are often forgotten. This system has safety-critical issues and the Start System use case includes a specification of all the safety checks that must be performed when applying power to the system. You don't want to accidentally start a conveyor that has service personnel working on it!

The Route System Maintenance Technologist (RSMT) is the person responsible for creating or predefining routes later used by the CRO. We can add new equipment to the system, define new kinds of equipment and define the characteristics of products carried by the system. The port facility deals with different types of iron ore coming in from a number of different mines, with different granularities and different levels of ore content. One of the things we have to be careful to avoid is loading the wrong product into a ship.

A word of warning about use case diagrams. The use case diagram is the tip of the iceberg. Don't start refactoring, reorganizing, and structuring the use-case model until you have written some
use-case specifications. The use-case specifications form about 95% of the use-case model. The diagrams only give you an overview, a summary of what is in the overall model.

As I mentioned earlier, use cases describe "flows of events". Figure 6 pictorially shows various kinds of flows and how these are represented in a use case specification.

**Figure 6. Basic and Alternative Flows**

The thick blue line running top-to-bottom is called the "Basic Flow". This describes what happens when everything goes right. There are various "Alternate Flows". A digression to describe the handling of an equipment fault is a good example of an alternate flow. Another example of an alternate flow would be a description of what happens when an operator cancels a previously requested action.

An important structure here is the Subflow. A subflow is like a subroutine call. We try to keep the basic flow simple and easy to read, and without subflows that can be very difficult to achieve. We use a subflow, for example, to describe in detail what happens when we start or when we position equipment in this system. Each of these processes is quite complex in itself, and if we put that detail in the basic flow, it would make the basic flow long and difficult to understand.

**Example basic flow describing the starting of a route**

Note: listed items in **bold** denote a glossary item, whereas list elements ending in an * symbol denote an **Alternate Flow Indicator**.

1. The CRO directs the System to start a previously **Selected Route**
2. The System determines the route is a **Valid Route**.
3. The System checks the **Starting/Positioning Strategy** for this Route and determines that the Route is not configured for **Starting/Positioning Strategy 2.1**.
4. The System determines that the Route will be a **Valid Active Route**.
5. The System determines that the Route Start Command is valid.
6. The System makes the Route Active.
7. The System determines that the conditions listed in Special Requirements, Section 3.1, "Conditions that prevent starting," do not apply to the Route.
8. The System asks the **Power Load Management System (PLMS)** if it is permitted to start the Route.
9. The PLMS grants permission to start the Route.*
10. etc.

At step 2, the system checks whether the route is valid. What happens if the system determines the route is not valid? This is described in an alternate flow. In the use-case specifications we used footnotes to indicate alternate flows with the footnote text containing a navigable cross-reference to the relevant document section. Again, this reduces clutter in the flow being specified and makes it easier to read. Here I’ve used the “§” symbol to indicate the presence of an alternate flow. I’ve also highlighted in red, terms that are defined in the project glossary - a most important document.

In step 3, the system checks the starting strategy for the selected route. "Starting/Positioning Strategy 2.1" describes a staggered start, which requires that all the equipment be in position before you start the route. This starting strategy is described in an alternate flow. Ultimately, the system asks the PLMS if we are allowed to start the route. What must happen if permission is not granted is described in an alternate flow.

Next we see examples of an alternate flow, and of a subflow.

**Alternate Flow Example**

2.2.6 Pre-positioning required

1. The System advises the CRO that the Route must be positioned prior to starting.
2. Use case ends

**Subflow Example**

Note: listed items in **bold** denote a glossary item, whereas list elements ending in an * symbol denote an Alternate Flow Indicator.

15. The System determines that the entire Route is **purged**.*
16. The System performs the following subflows simultaneously:
   NaN Position exclusively-owned Positioning Equipment
   NaN Position shared Positioning Equipment
   NaN **Stagger-start** Route **Transport Equipment**
17. The System advises the CRO that the Route is running.
18. The use case ends.

The alternate flow specifies what must happen if, for some reason, this particular route cannot be started unless all positioning equipment is already in position. The System issues a message to the operator, advising that positioning equipment must be in position before we actually start the route. At that point, the use case ends.

The subflow example provides more detail on how we actually start a route. The System checks whether the entire Route is purged (no material on the conveyors). If so, the system simultaneously moves the positioning equipment into position, both exclusively owned equipment and equipment shared with other routes, and then we stagger-start all the transport equipment,
to avoid overloading the power station. When the equipment is all running, we tell the operator that the route is running, and the use case ends. Steps "a", "b" and "c" of step 16 also refer to subflows.

**Special requirements**

There are a lot of requirements, particularly with a real-time system, that do not belong in the use case flows of events. Usually, they are nonfunctional requirements that relate to performance, availability, and so on. Those that relate specifically to a particular use case are captured in the Special Requirements section of the associated use-case specification. Sometimes, for convenience, even functional requirements may be relegated to this section. Examples are shown below:

**Example 1. Requirements preventing starting**

If an Operator attempts to re-start a Route for which the wharf conveyor has been stopped because of burden on the wharf conveyor at or within the gross stopping distance of the shipbuilder, the system shall:

- Raise and alarm and
- Issue a message to the Operator

**Example 2. Adjustment of Feed and Speed on Conveyor Fault**

The System shall adjust conveyor speed and source feed rate in accordance with the following algorithm...

The first example could be an alternate flow in the use case but to make the basic flow easier to read, we simply say "The system checks whether any of the conditions listed in Special Requirements section xyz prevent starting". In this particular case, if we have material on the wharf conveyor and it has been stopped because the conveyor might end up dumping ore on the deck of the ship, we have to raise an alarm and advise the operator.

The second example is a detailed description of an algorithm that is used to adjust the feed rate and the speed of a conveyor. Again, the basic flow would simply say "Feed and speed are adjusted in accordance with Special Requirements section uvw"

**Structuring of use cases**

Structuring of use cases is an advanced topic. You could completely document the requirements for a system without using these techniques. Structuring of the use cases allows you to avoid duplication and to ensure that there is only a single point of maintenance of use case text. The important point about structuring is that you must avoid making the use case model, and the use cases, difficult to understand.

Structuring techniques are shown in Figure 7.
**Figure 7. Structuring Use Cases**

On the left, we have a use case with a basic flow, alternate flows and subflows. Subflows and alternate flows can be modeled as separate use cases and extracted into separate use case specifications. The alternate flow becomes a new use case that extends the original use case. The subflow becomes a new use case that is included by the original use case. Generally, an include relationship is used only where there is an alternate flow that is common to a number of use cases. This is how we ensure that we only have to write the alternate flow once, and there is only a single point-of-maintenance. Often, an extending use case is created when functionality needs to be added to an existing use case but you don't want to modify the original use case.

Applying these structuring techniques to our product transport system changes our operations use cases to look like Figure 8.

**Figure 8. Revised Operations Use Case Model**

These new use cases are referred to as "abstract use cases" because they don't have an actor and you can't invoke them directly. If I were to hide all those abstract use cases, you could still see what the primary function of this system is quite clearly.
The set of error handling use cases all start with statements such as "when the system detects an equipment fault" or "when the system detects an overload condition." The use cases then go on to describe what the system does in response to those situations.

At the top of Figure 8 I have extracted a couple of abstract use cases: one for starting a route and another for stopping a route. Technically speaking, I have broken the rules here because the included use case should be shared by or used by more than one use case. I have extracted them here because, if we had documented everything in Transport Product, it would be about 300 pages long. Extracting the included use cases allows us to have many people working on the different aspects of the system at the same time.

Supplementary specification

We also have a Supplementary specification for requirements that don't fit into any of the use cases. As for special requirements that relate to a particular use case, these are typically nonfunctional requirements. There is a balance between the requirements that you have in the use cases and the requirements that you have in the supplementary specification. When you are working with a real-time system, you will probably find that you have a lot more requirements in the supplementary specification than you would if you were developing an IT system. This is because there are always a lot of critical timing requirements in a real-time system that would be documented in the supplementary specification.

Examples of non-functional requirements that would be captured in the supplementary specification are:

- The PT System shall, upon notification of fault requiring upstream interlocks to be set to "False", set Upstream interlocks to False within 0.60 second.
- The PT System annual Availability (refer Glossary) shall exceed 99.954%.
- The PT System Maintenance Procedures shall comply with the Client's maintenance strategy for real-time PLC control systems.
- The PT System shall use the existing PMAC System for the CRO, PCO, and viewer interface.
- The portion of the PT System controlling route interlocking and directly connected to route equipment and interconnected systems shall be PLC based and, in the event of failure of higher level control within the PT System, shall be capable of safely operating any route currently starting, running, or stopping.

Functional requirements may also be included in the supplementary specification. While use cases could be written for some of these, they would not be very big. It makes a lot of sense in this situation to put such requirements in the supplementary specification rather than write a full use-case specification. Some examples are:

- The System shall advise the Operator if the sum of the source feed setpoints for a route is more than 10% less than the minimum capacity for the route.
- If a feed source has been enabled for a route with a cleanup destination, the System shall:
  - Raise an alarm
  - Re-raise the alarm every 10 minutes while the condition persists
• If during a ship hatch change, burden is on the wharf conveyor at or within the gross stopping distance of the shiploader, the System shall:
  • Raise an alarm
  • Issue a message to the operator
  • Stop the wharf conveyor
  • Issue a request to the SHLS to stop the shiploader boom conveyor

**Benefits of use cases**

Use cases provide the following benefits:

• Give context to the requirements, clearly showing why a system is needed
• Make requirements much easier to read and understand than traditional declarative requirements by placing the requirements in chronological sequences
• Make it easier to gain agreement with customers because they can read and understand them more easily
• Are useful as a basis for project planning, analysis, design, test and documentation

On the other hand, there is a couple of perceptions regarding use cases that are incorrect. The first of these is that traditional approaches to writing requirements may initially appear to be more rigorous and precise, particularly with the variety of use case guidance presented in current books and papers. This is not the case, and we have been able to write extremely precise and rigorous use cases on this project.

Another perception is that use cases are writable by anyone without formal training. The rules of writing traditional declarative requirements still apply. You are documenting what the system has to do from an external point of view. You are not documenting how the system does what it is required to do. The use case requirements that you write have to be testable, just as traditional requirements do. Newcomers often continue to apply old habits such as functional decomposition. Those with a development background tend to take an internal rather than an external point-of-view.

**Use cases throughout the project**

As I mentioned earlier, use cases are used throughout a project.

We have insufficient space here to discuss the criteria for allocating use cases to iterations. Suffice to say, the criteria was based on a risk assessment in accordance with RUP. Here is the allocation:

**Project planning**

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Start, Stop and Handle an Equipment Fault on an Independent Route. Maintain Equipment and Routes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration 3</td>
<td>Create Product Gap.</td>
</tr>
</tbody>
</table>

RM13: Practical application of use cases to a real-time system
Use case analysis

Use case analysis is the process of identifying objects, classes, class attributes and class operations. While the implementation on this project was not object-oriented, a mapping was used from the object-oriented analysis model into a procedural design model. Figure 9 shows a partial sequence diagram created from the analysis of the Start Route use case.

Figure 9. Sequence Diagram derived from the Start Route use case

Test planning

I highly recommend the June 2001 Rational Edge article by Jim Heumann entitled "Generating Test Cases from Use Cases". This was the approach used on the PT project to identify test cases. Iteration 2 test cases for the Start Route use case are shown in Figure 10. Note the linkage between test cases and use case requirements. This facilitates ensuring that all requirements are tested and allows the test cases impacted by requirements changes to be identified.
Figure 10. Test Cases for and derived from, the Start Route use case

Lessons learned

1. Focus on externally visible behavior. I can't emphasize this enough, which is why I'm repeating it here. On this project, I used to tell the use case modelers regularly to imagine they were the customer flying over the port facility and ask themselves, what they wanted to see happening. If you write a requirement describing something that is not visible, you can't test it.

2. Do not introduce design. This is related to lesson 1. Here's an example:

Each route being fed from the Source "owns" one feed source "Enable" signal. The "Enable" signals are OR'd internally to derive the physical signal to the feed source. (Use case then written in terms of internal signals)

There are a number of reasons why you shouldn't do this. Firstly it makes testing difficult because the behavior described is not externally visible. Secondly, if you put design information in the use case, you are required to comply with that because it's a specification, so not only do you have to demonstrate compliance with it, it also constrains your designers.

3. Don't write use cases for internal system monitoring processes. In this particular system, which is a real-time system, there will be a lot of processes running inside the system that check things and monitor things, polling equipment, and so on. You shouldn't describe this in your use cases because again, it's design - it represents just one possible implementation.

Unless the process is related to the purpose of this system, such as a "Monitor Building" use case in a building monitoring system, what you should be doing here is writing requirements in the supplementary specification. For example:

- The system shall check the status of all equipment every 200ms or better still,
- The system shall initiate response to equipment failure within 200ms of the failure. And,
• The system shall ensure that the product blend ratio is within +/-5% of that set.

Alternate flows in use cases, or extending use cases, would then describe what happens if the system detects a problem. These requirements might drive you to design processes that poll equipment or the equipment might generate interrupts, but these are design considerations and don't belong in the requirements domain.

4. Don't "freeze" the use cases too soon. Freezing the use cases too early in a project causes a lot of problems because, as you and your customer understand the requirements better, you will find opportunities for restructuring the use cases or you will need to introduce new use cases or you will make changes to existing use cases. Requirements changes will occur, but they need to be managed within the constraints of an iterative process with proper impact assessment and associated schedule and budget changes if appropriate. If you are using an iterative process, you at least get the opportunity to get things right in a later iteration.

5. Don't let pursuit of the perfect use-case model blow out the schedule. This is a corollary to lesson 4. You will never be able to get a perfect use-case model. At some point, you have to stop tinkering with the use-case model and actually develop the system! It is possible to develop a system that meets the customer's requirements even though the use-case model might not be perfect. With this lesson in mind, Figure 11 shows what the PT use case model actually looked like.

**Figure 11. The Imperfect Use-Case Model**

You will see that Select Route, Start Route and Stop Route are primary use cases. There is no overarching Transport Product use case to tie all of these other use cases together, and we had to write requirements in the supplementary specification to cover all that. The operator or stakeholder derives no value from simply selecting, starting or stopping a route. The primary reason the use-case model is imperfect is because the requirements had to solidify at some point so that the developers weren't continually facing a moving target, and a system could actually be developed, tested and deployed.
6. Don't be afraid to capture the detail. Use cases are supposed to be easily read by everybody, but that doesn't mean you can take somebody off the street and give them a complex use case describing a complex system and have them understand it.

Questions to ask yourself are: How much detail does your customer want? How much latitude do you want to give your developers? If you don't have a lot of detail in the use case, you leave the developers open to make decisions themselves. Do you want that? Your developers may have extensive domain experience and you're happy with that. On the other hand, if you are contracting out the actual development, it is unlikely that inadequate detail would be acceptable. You need to fully describe what is important to the customer while providing sufficient detail for development, test and documentation.

If there is a large amount of detail, look at moving the detail to the Special Requirements or Supplementary Requirements area, or supplementing the use cases with Activity Diagrams. (See lesson 9)

7. Don't introduce explicit use-case interactions. Use cases don't interact directly. They might influence each other but only by virtue of manipulating the system state.

8. Don't build architectural decisions relating to resolving resource contention issues into the use cases. This is related to lesson 2. For example, what happens when Operator Enables Feed on a route while a second route sharing the same source experiences a fault requiring the source to be disabled?

These situations are described in different use cases. In each use case, you simply describe what you want the system to do under a given set of circumstances. In this case, how the system internally works out what signal to send to the source equipment is an architectural or design issue. The system architecture needs to resolve issues such as ensuring desired behavior when signals are received very close together, in various orders or very frequently.

9. Use Activity Diagrams to make complex use cases easier to understand. Activity Diagrams can be created as the use case specifications are written. These are often synergistic activities.

**Summary**

The use of Use Cases for specifying the PT System has been of great value to the project. Use cases should be considered for system specification whenever there is significant externally visible system behavior.

For any system, when identifying use cases, focus on the goal of the system and the value it provides to stakeholders, and when writing use case specifications maintain an external view of the system.

For real-time systems, expect a greater percentage of the system requirements to reside in the supplementary specification.