Methodology for developing Use Cases for large systems

Arnon Rotem-Gal-Oz

Preface

It was back in 1986 when Ivan Jacobson first invented Use Cases [Jacobson 2003]. It took several years for use cases to evolve until the definition, used today, was refined: "A description of set of sequences of actions, including variants, that a system performs that yield an observable result of value to an actor" [Booch 1999]. There are quite a lot of books, papers and other resources that explain what use cases are and how to go about modeling a use case model – however, the simplistic examples and techniques detailed in many of these resources either don't deal at all or don’t scale-up to deal with the non-trivial, real world, challenges of building a use case model for modern large and complex systems.

The purpose of this paper is to offer a methodology for creating and building a Use Case Model that caters for the needs and challenges of large and complex projects. The paper will explore some of the major challenges that are typical of large projects and will demonstrate practical steps for mitigating these challenges. To clarify - this paper is not about explaining why use cases should be used for enterprise requirements analysis nor is this paper about the mechanics and process of writing the single Use Case. The paper is about the Use Case Set and about the process to achieve a robust set that can serve as a good foundation for designing and building large software systems.

The paper begins with an overview of the naïve use case modeling process, followed by the challenges posed by large projects. This is followed by an overview of the suggested methodology, which is then detailed step by step.
It is important to show that the methodology elaborated in this paper is rooted in practical experience, thus I'll try to illustrate the key issues through examples. However, It is somewhat problematic to come up with a suitable example – On the

one hand an example that is too simplistic will miss the main purpose of the article (large, complex systems…) ,and on the other hand a complete real-life example will be too long and will cause the methodology to be lost in the details. The solution to this dilemma is to use examples that are small, local, snap-shots of the requirements of a single large system. The example chosen is a command and control system for police forces. The project described here is fictitious but it is closely related to a real project with similar size and complexity. The background needed to understand the big-picture of the example is described in Appendix A.

Introduction

The use case model describes the behavioral view of the system. The main artifacts of the use case model include:

- Actor list – A list of all the actors found and their relationships – maintaining an actors list is important especially when there are a lot of actors and there are several teams working in parallel to help prevent the teams from identifying duplicate actors within the model (more on that later).
- Use Case Packages – represent the contextual hierarchy between the use cases. This artifact contributes to the differentiation of the different use case levels as well as enhancing readability by grouping of use cases by subject. Use case packages can also be used to divide the work between the different teams.
- Use case diagrams – The diagrams are the graphical/pictorial representations of the use case model. Besides visualizing the interaction between the actors and the use cases, their more important role is describing the relations between the various use cases in the model.
- The use-cases text –Word documents containing the use cases. In most cases the use cases are written according to a template set by the projects/organization.
- Use case views – Several views that help understand the model from different angles (of the different stakeholders).

It is important to remember that use cases do not capture the entire requirements of the system – e.g. non behavioral aspects of the system such as performance, security;

environment constraints (such as specific OS, Hardware etc.) are not captured and should be elaborated separately. Some of the non-behavioral aspects can be captured within use cases - if they manifest themselves in the context of the use case, e.g. a performance requirement for response time can be captured when describing the "Handle Emergency Call" use case of the Emergency Communication Center.

The naïve process for building a use case model is very straightforward [Armour 2001]

1. Find Actors
2. Find Use Cases
3. Describe the Use Cases

The problem with this approach is that such a simple process just doesn't cut it when it comes to large and complex systems.

Most of the challenges to the naïve process posed by large/complex systems are indeed related to size – either of the requirements gathering team or the model itself.

One of the major problems with a large use case set (which is common for a large system) is that there is a good chance that the model is inflicted with duplicates (more on that later) and more importantly inconsistencies between the different use cases – starting from boundaries mismatches and ending in contradicting use cases.

Another of the challenges related to the use case set is the problem of use case explosion, or simply put, too many use cases in the model [Lilly 2000]. The issue is, actually, a little more subtle as there are multiple levels of use cases and the problem can occur at each level separately. When a use case set has too many use cases they usually don't describe user goals but rather trivial interactions or incidental actions of the users.

Another problem related to the use case set is making sure the requirements detailed in the model are "good" - as described in the "IEEE Recommended Practice for Software Requirements Specifications", i.e., Correct; Unambiguous; Complete; Consistent; Ranked for importance and/or stability; Verifiable; Modifiable and Traceable [IEEE 1998]. The problem of eliciting good requirements is, of course, not specific to large systems, but it is much more problematic when dealing with a lot of requirements.

Using large teams for the modeling effort can basically cause two major problems. One is that too many people working together (on anything) is both inefficient and leads to a lot of compromises (to satisfy the many different views) resulting in a "design by committee" effect [Adolph 2003].

The other major problem caused by large teams, dubbed by Donald G. Firesmith as the "Humpty Dumpty effect", is caused by dividing the work between many teams thus risking multiple, redundant, partial variants of the same classes and objects fragmented over many use cases. The result of that scattering of objects may be that " all the kings designers and all the king's coders are unlikely to put the objects and classes back together again without a massive expenditure of time and effort" [Firesmith 1996].

The other type of challenges is challenges related to the process of building the model. People (esp. developers) are drawn into writing the details early [Adolph 2003] - this has several negative effects, such as wasting energy on detailing requirements that will almost surly change when more will be known about the system (as the analysis progresses). Also, this hinders the possibility of achieving a coherent use case set that can be validated early by the stakeholders/customers.

Another process related problem (which is related to the "use case explosion" problem mentioned earlier) is of knowing when to stop [Adolph 2003, Firesmith 1996] or rather not knowing when to stop, which can lead to both, a lot of use cases and the use case building effort never to finish [Lilly 2000].

The last major problem[1] related to the process is trying to cover all the system requirements in one single pass [Armour 2001, Adolph 2003] this can have dire consequences such as large delays in the project schedule or even worse finding out a lot of that extra time was wasted as requirements change (when the project's team has better understanding of the system).

It is clear that the issues and challenges mentioned above cannot be solved by the naïve process mentioned earlier. The use case model building process should be

[1] There are many other potential use case related problems - both to writing the use cases themselves and to use case modeling in general - but they are not unique to large and complex system and thus, not in the scope of this paper

extended in order to mitigate these challenges. The following section details such an extension of the process to support building a model for large and complex systems. The suggested process for building a use case model – can itself be summed in a pseudo use case form:

Use Case: Build a Use Case Model

Level: Summary

Scope: The requirements engineer(s) build a use case model of the functional requirement for the project at hand.

Primary Actor: Requirements Manager

Supporting Actors:

- Requirements Engineers
- Architect
- Customer

Stakeholders and concerns:

- Project Manager: Wants to understand the requirements and assess the development effort.
- Architect: Wants to understand the breadth and scope of the problem, and the customer's and end-users needs (in order to design a suitable solution).
- End-Users: Wants to make sure all their needs are addressed
- Customer : Wants to make sure the project team understand the requirements and that she'll get a viable product
- Development team: Wants to understand what they need to build.

Preconditions: None.

Success Guarantees: The project team has a good, viable description of the functional requirements of the project. The customer and contractor both understand and agree on the requirements.

Trigger: The Requirement Engineer decides to start collecting and eliciting the customer requirements.

Main Success Scenario:

A. Initialization Steps

0. The Customer described the problems and needs
1. The Requirements Manager and Architect define the system boundary

2. The Requirements Manager organizes the team (requirements engineers)
3. The Requirement Engineers build the Problem Domain Object Model (PDOM)

B. The Process

4. The Requirements Engineers identify actors
5. The Requirements Engineers identify use cases
6. The Requirements Manager organizes the model
7. The Architect prioritizes the use cases
8. The Requirements Engineers describe the use cases
9. The Requirements Engineers refactor the model

C. Supporting Steps

10. The Requirements Engineers Verify & Validate the model
11. The Architect identifies future requirements

D. End Game

12. The Requirements Manger decides when to stop

Variations:

1. Steps 5 and 6 can be done in parallel (i.e. find some use cases, organize them, then find some more etc.)
2. Steps 10 and 11 can (and should) be performed in parallel to steps 5-9
3. The process is iterative in nature, thus steps 4-11 (and 3, if needed) may be repeated several times with increased detail in each iteration.
4. In addition to Variation 3 – In certain situations it may even be needed to refine the system boundary/scope (step 1) after the first iteration of steps 4 and 5.

Building a Use Case Model

Initialization Steps

Step 1: Define the system boundary

"A journey of a thousand miles begins with a single step" - Though it may be tempting to jump in and start modeling the use cases right off, it is very important to make the appropriate preparations and decisions before you begin.

The first thing to do is establish an early vision of the system (which will be updated later as we gain more understanding of the system). This is an important step since the vision captures the essence of the requirements (the fundamental "why's and what's" of the project) – thus, having a clear vision increases the chance to develop a system that will meet the stakeholders real needs [Probasco 2000].

The vision (& scope) statement should address the following issues [Armour 2001, Probasco 2000]:

- What problem(s) are we trying to solve?
- Who are the stakeholders? Who are the users? What are their respective needs?
- What are the main goals of the client organization? What are the main goals of the department/unit the solution is built for?
- What are the main goals of the system?
- How would this solution affect our (the contractor's) business?
- What are the boundaries of the solution? What are the major functional and non-functional requirements?
- What are the future directions of the product?

Example 1: Police Force Command & Control – Vision document

The vision document, as mentioned above, provides an overview of the problem; Who are the stakeholders that have vested interest in the project; the highlights of the solution; etc. The text below shows few examples for topics that are part of the vision statement for the Police Force Command & Control project.

Problem Statement (Excerpts)

The problem of	The emergency call center procedures are manual and take long time to accomplish. The time it takes from accepting a call to a police car arriving at a crime-scene can get as high as 30-40 minutes
affects	Citizens
the impact of which is	The police, sometimes, arrive too late, which is both, well, too late and makes it is harder to apprehend the offenders.
a successful solution would	Decrease the response time for on-going calls/cases to 15 minutes or less.

The problem of	Police car location is based on the policemen calling in their location by radio and by an operator manually marking their whereabouts on a printed map – this makes the locations both outdated and inaccurate.
affects	Citizens, Policemen
the impact of which is	Cars that are close to the whereabouts of the crime – are not used – which increases the time it takes to arrive at a crime scene.
a successful solution would	Allow better utilization of available beat cars for on-going calls response by a factor of 4 (or better)

The problem of	Police cars are not serviced on a regular basis, the registration of police cars and their management is a manual operation.
affects	Logistics Commander, Policemen
the impact of which is	Cars break-down in mid-operation, fuel costs are high.
a successful solution would	Decrease the number of unexpected beat car break-down during operation by a factor of 2

Stakeholder Summary (Excerpts)

Name	Represents	Role
Citizens	While not a part of the system, citizens are the underlying financers of the system (via tax money). Even more importantly, citizens will benefit from a more efficient police operation by getting their emergency calls treated more quickly. Also they will benefit from lower crime rates.	The citizens do not play a role in the development process.
Customer Representatives.	Customer	Ensure the cost and scope of the project will stay on-track
User Representatives	End-users	Ensure that the system interaction design will benefit the end-users.

User Summary (Excerpts)

Name	Description
Watch Commander	The watch commander is the commander on-duty in every watch (there are 3 watches a day). She is responsible for the overall operation of the district.
Beat Officer	Any policeman that is part of a beat team – these police officers patrol in their cars in a specific beat during a specific watch.
Logistics manager	The logistics manager is responsible for all the equipment including police cars and the various detectors (camera, radars etc.)

Summary of Capabilities (Excerpts)

Customer Benefit	Supporting Features
New Emergency center operators can quickly get up to speed.	Wizards ,easy to navigate maps and concise forms assists operators in quickly identifying caller, available police cars for dispatch etc.
Watch commander job is easier and improved because data is accurate, up-to-date and nothing falls through the cracks.	Tools supporting situation awareness updates automatically; alerts on problems are automatically generated; statistical reports, as well as, detailed reports are generated on request.
District chiefs and the police chief can identify problem areas and gauge staff workload.	Trend and distribution reports allow high level review of problem status. Incident analysis (including spatial analysis) tools support identification of problem patterns.

	Logistics Manager job is more efficient and equipment maintenance is better.	Usage is tracked automatically as well as service history for all equipment types; Reports are generated by the system for equipment that should be services on a time/usage basis.
	Beat cops job is easier and more efficient.	Integrated navigation system improves response time for calls; on-line access to license and registration databases improves tracking and identifying offenders.

Many of the answers to the questions in the vision statement serve as helpful, relevant information for the use case modeling effort. For example the *main users* will help us identify actors of the system; *future directions* will help us define change cases etc. The most important answer[2] however, is the definition of the system boundary. The system boundary is important because the core of the use case approach is to define the system from the point of view the entities which are out-side of it. If what is inside the system and what isn't is not known, the modeling effort cannot really begin. Another point is that a document with a shared clear vision [Adolph 2003] helps coordinate and focus the requirements teams (and later the designers and coders) on the same goal.

While, this may not be directly related to the use case effort – it should be noted that in large systems the vision statement is likely to be refined through the early stages of the project ("inception phase" in RUP terms). These refinements can include the functional requirements (along with measurable acceptance criteria), financial constraints, trade-offs etc.

Step 2: Organize the team.

The second step to carry out before starting the modeling effort itself is an organizational one. – The teams (sizes and structure) that will be involved in the modeling effort should be determined.

The first aspect regarding teams is the size of the writing teams and the number of total writing teams – in both cases it is desirable to keep sizes as small as possible [Adolph 2003]. Having a small team (2-3 people) write any individual use case will increase the efficiency of the writing process as well as help mitigate the risk of feature bloat. Although it is tempting to throw a lot of people at a large problem (There are a lot of requirements to cover in large projects) -having a few teams as

[2] in the context of the use case model

possible will help maintain the model's consistency (maintain conceptual integrity) and, yes, even get to the target faster - as the following anecdote told by Fred Brooks demonstrates[3]:

"It is very humbling experience to make a multimillion-dollar mistake, but it is also very memorable…The architecture manager had 10 good men. He asserted that they could write the specifications and do it right. It would take ten months, three more than the schedule allowed. The control program manager had 150 men. He asserted that they could prepare the specification, with the architecture team coordinating; it would be well-done and practical, and he could do it on schedule. Furthermore, if the architecture team did it, his 150 men would sit twiddling their thumbs for ten months. To this the architecture manager responded that if I gave the control program team the responsibility, the result would not in fact be on time, but would also be three month late, and of much lower quality. I did, and it was. He was right on both counts. Moreover, the lack of conceptual integrity made the system far more costly to build and change, and I would estimate that it added a year to debugging time." [Brooks 1995]

The second aspect is the composition of the teams – while the teams should be small they shouldn't be homogenous. The teams should be staffed with people having different specialties (including domain experts) [Adolph 2003] - it is especially important to have customer involvement in the process, including if possible end-users. Balanced teams will help keep the use cases at a level understandable by all the stakeholders (including the development teams).

When developing large projects, the number of concerned parties (external stockholders, domain experts, annalists, developers etc.) is usually large as well; this leads to a delicate issue regarding the size of the teams versus balanced representation by using heterogenic teams. In order to solve this issue it is suggested to use two tier reviews[Adolph 2003], that is, first letting the (small) team agree on the use-case, then holding one or more reviews to agree on the use-case on a broader forum and at the end have a single large review where the use case will be presented (along with the other use cases) to all the concerned audience.

Another team related issue in large projects is preventing overlaps and inconsistencies between the works of the various teams. In order to balance this effect a single person

[3] The storey in mythical man-month is not about a use cases effort but the same principal holds.

should be responsible for coordinating the whole process (a "requirements manager"). This person's responsibility would include dividing the work between the teams and also serve as a first tier reviewer to maintain the consistency of the overall use case model as well as minimizing the chances for overlaps[4].

The vision statement, described earlier, will help the team focus on the system goals rather than on the (lesser) goals of the parties involved in the teams.

Step 3: Build a Problem Domain Object Model (PDOM)

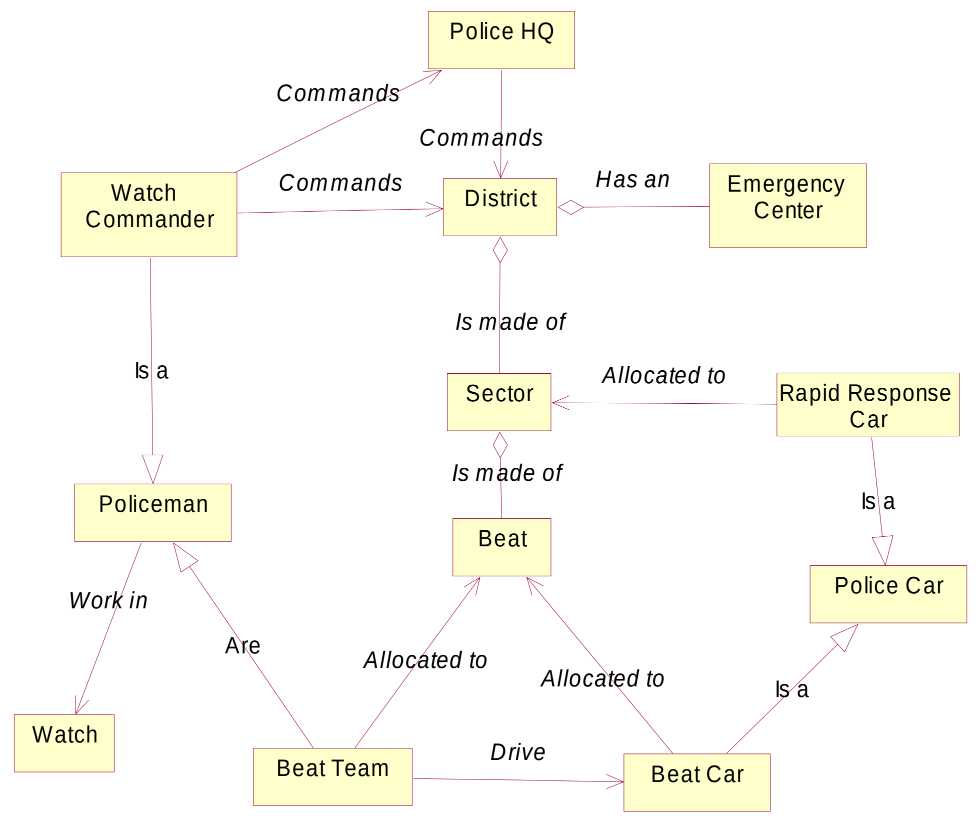
Another important artifact that helps in the coordination of multiple teams is the PDOM ,also knows as Domain modeling, which helps in defining the system boundry and, more importantly (since the boundary is also defined in the vision document) set a common vocabulary [Armour 2001, Rosenberg 2001] for the different teams to use amongst themselves and while talking with end-users or any other customer representatives.

While the Use Case model is part of the dynamic view of the system, the PDOM is part of the static view. The PDOM is, essentially, a class model of the real-world things and concepts related to the problem the system is designed to solve and that the system must know about [Jacobson 1992].

[4] This, and additional issues related to the quality of the model will be discussed later as part of the discussion on verification and validation

Example 2: excerpt from Police Force Command & Control PDOM

The PDOM is usually a set of UML object diagrams depicting the relations between the various objects in the problem domain, followed by an explanation of the terms/objects identifies in each diagram. The example below shows a small part of the police Force Command & Control PDOM. The diagram does not contain all the relations but it sufficient for understanding the PDOM's structure (The data in this example is based on [Chicago 2003, O'Connor 2003]).



- Police HQ – The headquarters responsible for all the districts in a municipality.
- District – A geographical division of a municipality. A municipality can have up to 25-35 districts depending on the size of the city
- Sector – One of three geographic divisions within a police district, comprising three to five beats
- Beat - A geographic area assigned to specific officers for patrol
- Watch - A police shift. The police workday is divided into three watches. The

first watch begins at 11 pm or midnight; the second, at 7 or 8 am; and the third, at 3 or 4 pm.

- Beat Team - The eight or nine officers from all three watches assigned to the same beat, and the sergeant who serves as team leader
- Beat Car – A police car assigned to a specific beat.
- Rapid Response Car - A squad car assigned to patrol a sector within a district and respond to in-progress (emergency) calls
- Watch Commander – A lieutenant or captain who directs all police activities within a district during a specific watch. Examples of the watch commander's duties include deploying patrol officers within the district, approving arrests, and checking the status of the lockup
- Emergency Communication Center – A 911-like call center for processing emergency calls.

The PDOM is not part of the use case model, the reason it is mentioned in this paper is that it serves as a balancing factor for the use case model [Jacobson 1992]. First and foremost, it helps mitigate the risk of ending up with a functional (structural) model instead of an object model [Firesmith 1996]. This can be achieved by writing the use cases in the context of the object model [Rosenberg 2001] i.e. also take into account the objects in the problem domain and not look at the use cases from a pure external actors viewpoint.

The PDOM serving as a vocabulary of system concepts and objects, already mentioned for helping teams communication, also helps to achieve greater consistency between the use cases themselves [Armour 2001].

The PDOM can be developed iteratively, as the requirement gathering effort progresses [Armour 2001]. Building the PDOM iteratively in conjunction with writing the use cases can be used to help set the detail level of the use cases.

It is important to remember that the PDOM is not the system's class model. The PDOM is just one of the 3 class model that are, usually, built in an orderly development cycle. The additional two are the analysis class model (which uses the

PDOM and the use case model as a source) and the third is the design class model (i.e. the actual classes used by the application).
The PDOM serves several other aspects of the development process (that are not in the scope of this paper) for example it serves as a source for candidate classes for the design, to help define the system interfaces, etc.

The Process

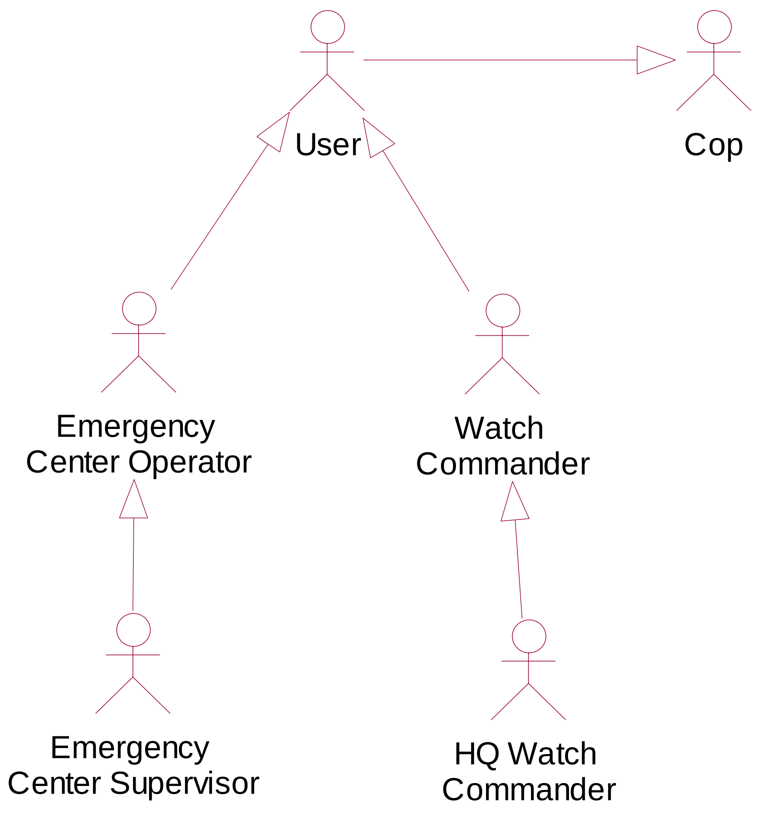
Step 4: Find actors

Actor is a role that a user or external system plays with respect to the system under design. Usually finding actors is not a goal in itself, but rather a good starting point from which to identify use cases. There are some situations where it may be beneficial to track actors, for example: if the system will need to be configured differently for various kinds of users. The use case performed by each actor, in this case, makes the usage profile within the system [Fowler 2000].
Finding actors is not a specific task for large systems - it is a recommended task for any use case modeling effort. In the context of large systems, it is important to remember that when starting out, the goal is to identify the major actors of the system and there is no need to make an exhaustive list of all the actors [Armour 2001]. Additional actors will be reviled as the level of detail of the analysis increases.
The main source for the list of actors is interviews and brainstorming with end-users and customer. Other resources include documents, such as [Armour 2001]: organization charts, written specification (if there was an RFP), manuals of current processes and systems etc.
An important issue is the distinction of roles and job titles. It is very convenient at the beginning of the use case modeling effort to identify primary actors as the job titles used in the organization. The only problem with that is this is usually not true – A single person may play several roles when interacting with the application, for example any person (operator, Watch supervisor etc.) in the Emergency Communication Center may pick up the phone and register an incident (not just operators). The process then, is to look at the different job titles and identify the roles they can play. To solve this issue it is best to maintain a job-titles/roles matrix, this way the business context of the roles is not lost and the real roles are used in the use

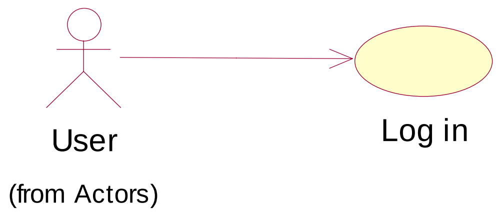
cases [Cockburn 2001]. This relation can also be shown in Actor diagrams using the generalization relation

Example 3: Actor's generalization

The diagram below shows a small fraction of the Force Command & Control identified actors along with (some of) their relations.



This example means that when a reader looks at a use case like the following:

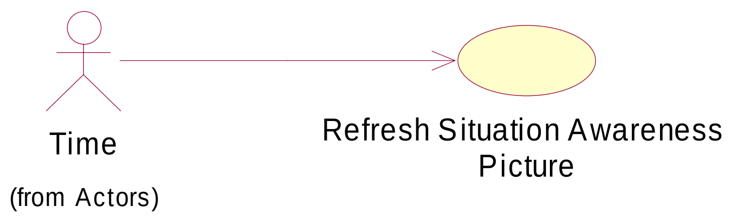


She can understand that, any actor that derives from user (Watch commanders, Emergency Center operators and their supervisors in this example) can log in the system. Additionally, (from the first diagram) it can be understood, that while all the system users are cops not all the cops can log into the system (since a cop is not an actor that logs in).

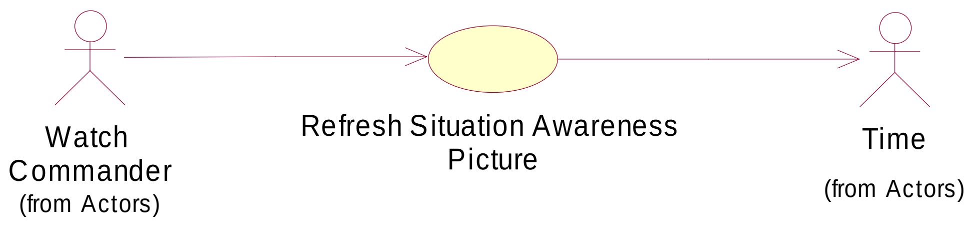
Another interesting issue regarding actors is the use of time or clock as an actor. Sometimes use cases appear to be triggered automatically based on a clock and it may seem appropriate to use the time or clock as the primary actor for the use case. The problem with this is that the time doesn't really have a vested interest in the system – there is always a real actor in the system that has interest in the use case and choosing to use the time as a primary actor hides her. The (recommended) alternative (to using the time as a primary actor) is to put the real actor as a primary actor and use the time as a secondary actor [Crain 2002]. This solution is better since both the fact that there is a real actor which is interested in the use case is preserved as well as the fact that time is a factor in the use case.

Example 4: Time as a secondary actor.

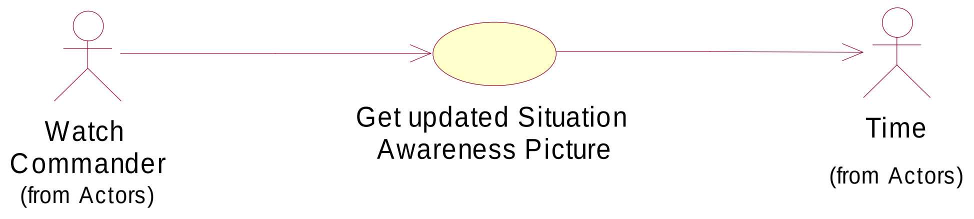
Looking at the time as an actor in the police Force Command & Control system - several use cases can be identified, for example: *Refresh Situation Awareness Picture*. This is a task that is done automatically every X seconds (as well as upon event, e.g. whenever there is a new emergency incident) and refreshed the location of police cars on a map (among other things).



Studying this use case more closely reveals that the real actor is the watch commander in charge of the district and not the time. The use case is then depicted as follows:



Analyzing this further, the real actor goal can be derived – which is not to refresh the situation awareness picture – but rather to get an updated awareness picture – so the final use case is:



Step 5: Find use cases

Finding use cases is, naturally, one of the most important activities in use case modeling. In the initial iteration, the idea is not to find all of the system use cases, but rather to find enough meaningful use cases that will give a good overview of the system. This overview can then be used to identify risk factors, formalize an initial candidate architecture etc. It is important to remember, especially in the initial phases that actually we are identifying use case candidates – not all of which will eventually be developed as use cases.

People, and even more so, developers are drawn into writing the details early [Adolph 2003]. This is especially true when people are overwhelmed by a large system and/or not sure how to proceed. This should not be encouraged, as it can (and usually will) lead to losing focus and losing energy as people get bogged down in the details. Also, too much detail early on, when the complete picture is not fully understood, will result in requirements that are volatile and likely to change later on.

Therefore it is best to follow, what Adolph et al call BreadthBeforeDepth pattern [Adolph 2003] and conserve energy for later stages.

It is very beneficial, especially in large projects, to develop the use cases iteratively. Developing the use-cases in one pass will, actually, delay the project (incur "water-fall development"), not to mention that the requirements are likely to change as more information is discovered and understood about the system. [Adolph 2003].

The same discovery techniques and guidelines apply to any of the iterations - where each iteration increases the depth, precision and accuracy of the use case set.

There are basically four ways for discovering use cases [Ham 1998]:

- Scenario driven
- Actor/Responsibility
- Unstructured aggregation
- Mission decomposition

Scenario driven

Scenario driven use case discovery is the traditional OO approach for use case elicitation. The approach is to examine the list of primary actors and look for questions such as [Armour 2001]:

- What measurable value is needed by the actor?
- What business event might this actor initiate (based on her role)?

- What services does the actor need from the system?
- What services does the actor provide?
- What information does the actor need from the system?
- What are the activities that are recurring and triggered by time?

Example 5: Finding Use cases

Each question type can help discover different use case types. The list below shows few example for use cases discovery by utilizing the above mentioned questions.

- What measurable value is needed by the actor?
  - Plan Special Op.
  - Monitor Special Op.
  - Analyze Crime Patterns.
- What business event might this actor initiate (based on her role)?
  - Handle Emergency Call
  - Call Car for Service
- What services does the actor need from the system?
  - Find Navigation Route
  - Get Unit Status
  - Map Incidents
- What services does the actor provide?
  - Dispatch Units
  - Issue Tickets
- What information does the actor need from the system?
  - Get Car Registration History
  - List Duties
- What are the activities that are recurring and triggered by time?
  - Get Updated Situation Awareness Map
  - Generate Emergency Center Statistics Report
  - Generate Crime Trends Report.

It is important that each of the use cases will encapsulate a meaningful value for the users (actors). Use cases (esp. at the higher levels) are about goals and not about user tasks – the difference is subtle but important- A goal is an end condition whereas a task is an intermediate process performed to achieve the goal; for example, at the initial stage, a use-case like *Handle Emergency Call* is a valid use case, while a use-case like *Fill-in Incident Form* is not.

Actor/Responsibility

A variation on the scenario driven elicitation is the Actor/Responsibility approach - Taking each of the actors, finding their roles, the responsibilities they have for accomplishing tasks and the collaborations the actors have with other actors (to accomplish the tasks). The use cases are discovered by identifying the productive task results [Ham 1998].

Unstructured aggregation

Unstructured aggregation is based on examining RFPs or RFP-like documents (traditionally consisting of "shall" statements) - any active verb requirement can be considered as a candidate use case. One benefit of this approach is that it helps incorporate non-functional requirements into specific use cases.

Mission decomposition

Mission decomposition is somewhat similar to traditional decomposition- it begins with a mission goal, which is decomposed by asking what need to be done to reach that goal (product, services, etc.) and this decomposition continues until a leaf can be considered as the output specification for a use case. The use case is elaborated by identifying the actors, events, business rules etc. that apply to that mission component [Ham 1998]. A benefit of this approach is focus on the main functionality (mission) of the system and not on "nice-to-have" functionality.

Going through the different discovery techniques will yield a handful of use-case candidates. One of the first things to do is to name the use-cases. The use case name should reflect the interaction with the system as precisely as possible – as it is viewed from the perspective of the actor. On the formal side- the common practice is to use an active verb + [qualified] object [Gottesdiener 2003] for the name e.g. "Navigate to crime-scene", "Plan Special Op" etc.). It is better to use precise verbs like "monitor", "notify", "approve" instead of vague names like "do" or "process".

Each new use case should have its actors listed (at least the primary one) and should contain a short description summarizing the business goal and/or the purpose of the use-case [Armour 2001].

It is also beneficial to determine the scope of each use case so that the boundary between the different use cases can be determined. Specifying the scope can be achieved by elaborating it in the text of the general description, a more formal way (which will also benefit the later stages) is to use specific fields for this information namely the preconditions, success guarantee and trigger. These three fields are used to show what should have happened before the use case, what happened in the use case and how the use case is started. Specifically:

- Preconditions describe what must happen before the use case begin – i.e. the state and/or status that the system will have before letting the use-case start [Cockburn 2001], usually the system will actively ensure these conditions (in other words – another use case would run and set the conditions up), though sometimes the desired state can be guaranteed via design decisions.
- Success Guarantee describes the state of the system after a successful run of the use case. This should include what interests of the stakeholders (including, of course, the actors) are satisfied after a successful conclusion of the use case [Cockburn 2001].
- Trigger describes the event that gets the use case started [Cockburn 2001]. It is usually performed by the primary actor.

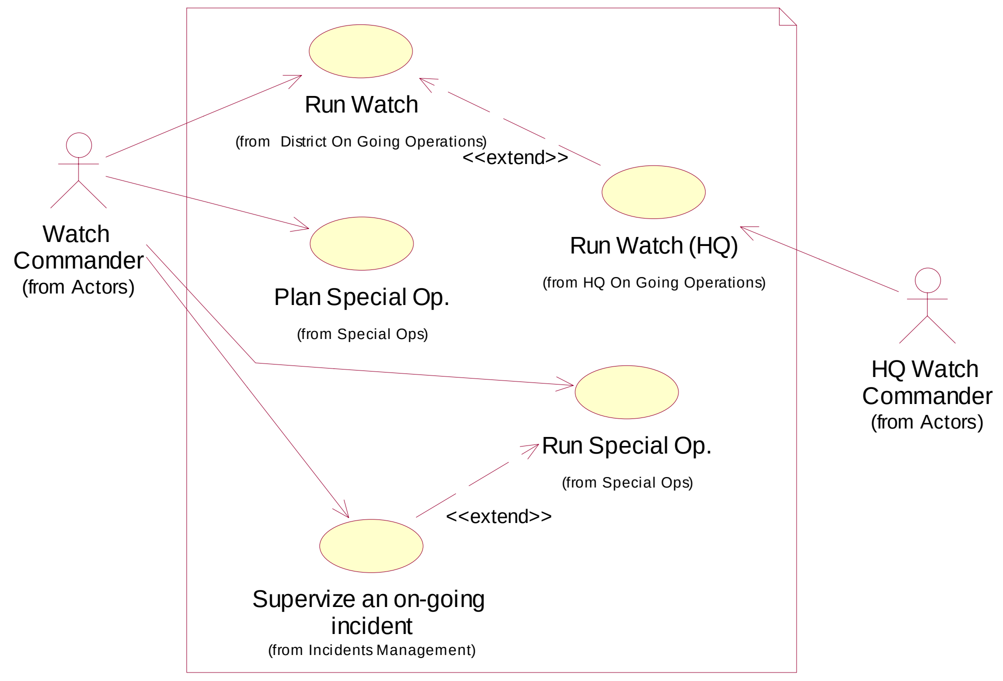
Lastly, the use case should be given a unique id - for model management purposes.

Example 6: Use case diagram

The Top-level use case model for the Force Command & Control system has twenty six use cases grouped into six categories (more grouping and ordering the model later). The following example shows the top-level use cases for two of the groups.

HQ Management

The following 5 use cases are the core use cases identified for HQ management



The first use case was hatched by examining the primary goal of the Watch commander

Use Case: Run Watch

ID: UC1

Scope: The Watch commander, having already logged into the system, initialize the watch by studying the summary prepared by the former watch commander (i.e. check out the open incidents; try to formalize situational awareness as to her forces etc.).

Next the watch commander supervise the on-going tasks of the watch.

Towards the end of the watch, the watch commander, will summarize the open incidents and significant events for the next shift.

Primary Actor: Watch Commander

Preconditions: Watch Commander Logged into the system

Success Guarantees:

- Watch status report had been prepared, saved and printed
- Watch statistics are saved in the system.

Trigger: The Watch Commander chooses to "Init. The Watch".

The other use cases where hatched by examining the commander's secondary goals, for example:

Use Case: Run Special Op.

ID: UC4

Scope: The Watch Commander chooses a Special operation to manage.
The task team chosen for the operation is briefed
The watch commander then monitors the operation as it unfolds (sending out orders as needed)
The task team is debriefed for the results and a final report is made.

Primary Actor: Watch Commander

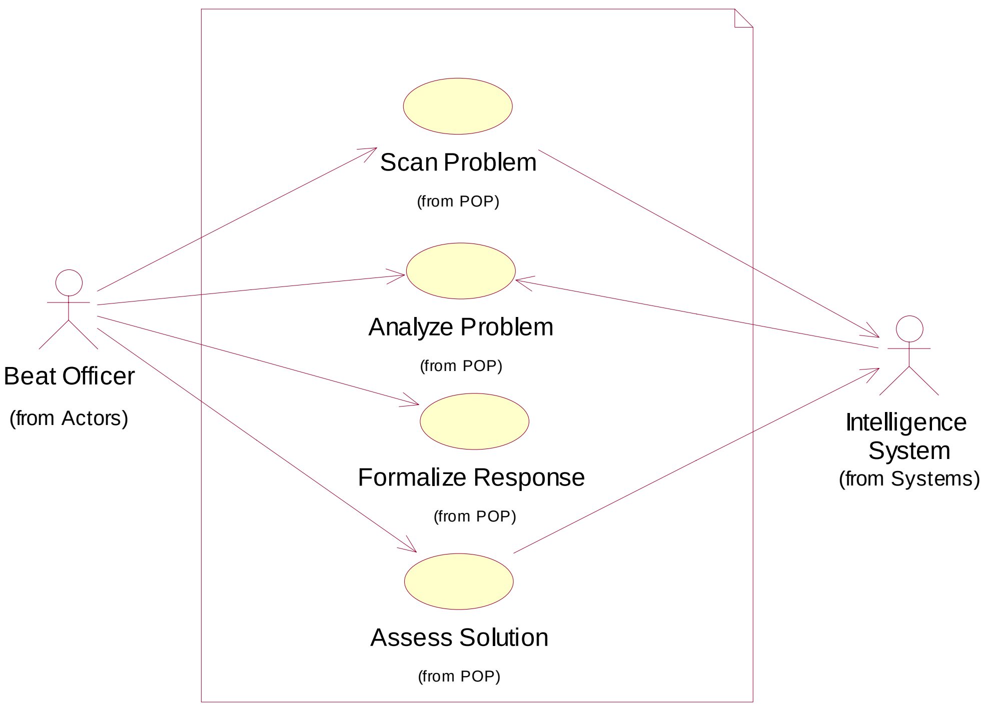
Preconditions: A Special Op. Plan is saved in the system.

Success Guarantees:

- The Special Op. recordings (Forces movement, Voice recordings etc.) are saved in the system.
- The operation's statistics are saved in the system.
- Operation Final Report is saved and printed.

Trigger: The Watch Commander chooses a Special Op.

Problem Oriented Policing (POP)
The following use cases are the core use cases for the POP sub-system. The use cases allow the actors to follow the POP methodology using the SARA process [Leigh 1996]: Scan, Analyze, Response and Assess.



Here is one example of the use case description from the POP group:

Use Case: Analyze Problem

ID: UC10

Scope: The officer searches for incidents related to the problem and analyze their characteristics (trying to find common patterns etc.) The system provides tools and data (using spatial analysis tools, data imported from intelligence system, data mining tools on the incidents statistics and details) on the following areas (in accordance with the Problem Analysis Triangle (PAT) methodology [Leigh 1996]):

- Features of the incidents location
- Features of the caller/victim
- Features of the offender / problem source.

This is an iterative process that stops when the officer believes she has a good understanding of the problem and it causes.

Primary Actor: Beat Officer

Preconditions:

- A problem has been identified and entered into the system.

Success Guarantees:

- The analysis data is saved in the system

- The analysis path is saved in the system (major decisions)
- The analysis conclusions are saved in the system.

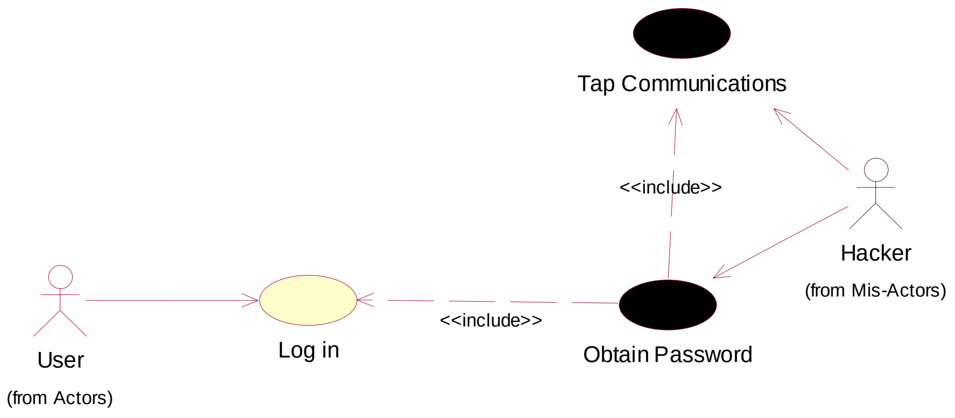
Trigger: The Beat Officer selects a problem to analyze.

An interesting issue regarding use case discovery is that of **mis**use cases [Alexander 2002] – simply put, a misuse case is a use case from the point of view of an actor that is hostile to the system and its (the use case) goal is a threat to the system (rather than a system function). Misuse cases are beneficial in eliciting non-functional requirements relating to security, safety, availability etc.

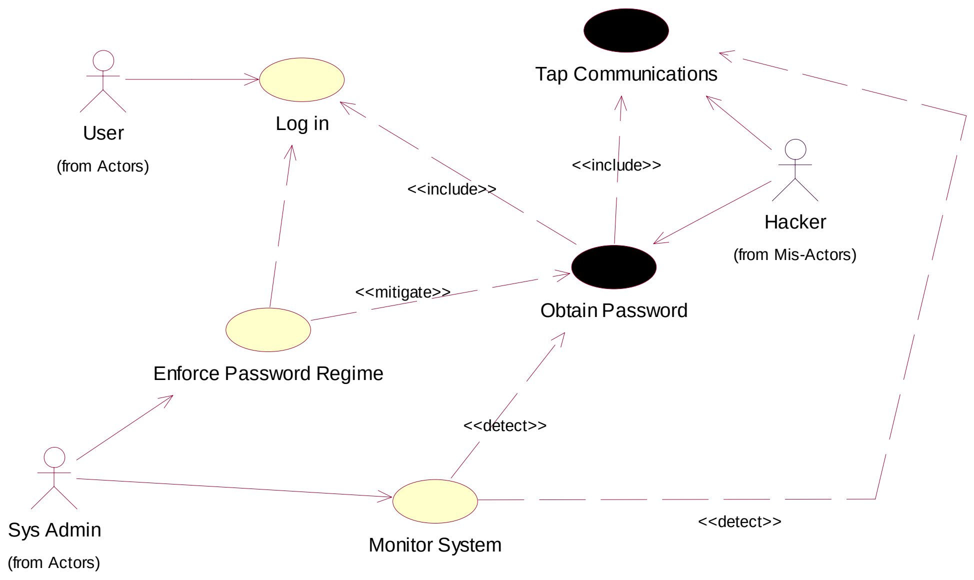
The idea is that once a misuse case has been identified (and later elaborated), analysis can be made to discover the real system use-cases that are needed to mitigate and handle the misuse case risks. Trade-off analysis should be carried out to determine if mitigating misuse cases risks is worth-while (from the cost and effort perspectives).

Example 7: Misuse Cases

The following example shows a misuse case and the analysis that follows (i.e. the new use cases added to the system in-order to take care of the problems raised by the misuse case)



The first diagram shows a hacker that manages to obtain password and log into the system by tapping the communications.



The second diagram shows two new use cases that are added in order to mitigate this threat.

- Monitor system – The system administrator will need to have a monitoring mechanism that will allow her to detect and trace any tapping or hacking activities.

- Enforce Password Regime – The use case will help make sure those passwords are both non-trivial and changed periodically. Which, in turn, help make the system less prone to break-ins. This use case is initiated by the system administrator but it also affects the log-in use case for the users.

Note that the Tap Communications misuse case is a source for several other misuse cases for the hacker such as "Monitor police moves" or "DOS/DDOS attack" etc.

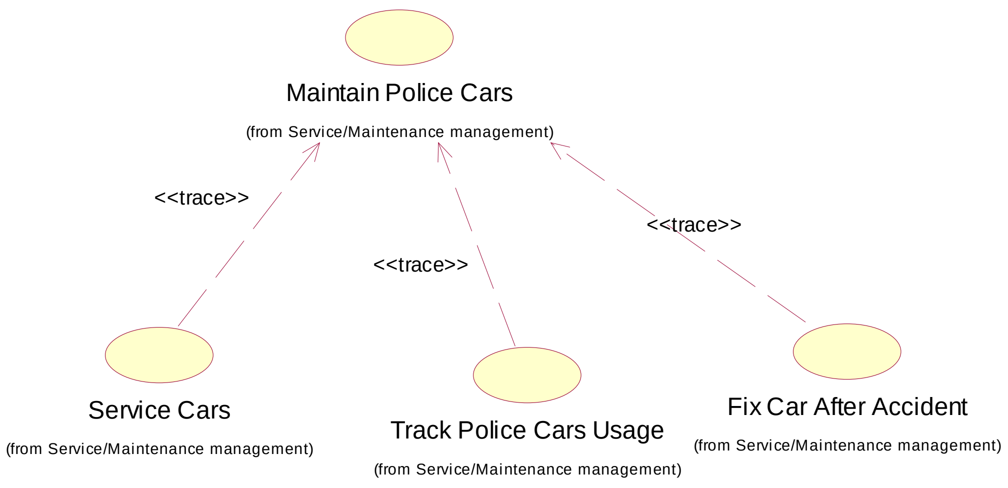
Step 6: Organize the model.

Organizing the model may seem a trivial task – it is, however, an important step, that contributes a lot to the manageability of the model. Readers (and writers) need an easy way to navigate the model and more importantly to understand it. Organizing the model can help determine the requirements applicable to the different aspects of the development and help identify inconsistencies or overlaps [Gabb 2001]

The simplest form of organizing the model is by level of detail or what is called "everUnfoldingStory" [Adolph 2003]. The idea is that the use case set is a hierarchical story that can be unfolded for more detail or folded up for higher level view (and more contexts). This organization principle means that each diagram will only show 1 or 2 levels of use-cases, where the lower level use cases are related to the upper level use case by an include or trace relation. Navigation is as easy as asking "why?", or "what for?" to navigate up and "how" to navigate down.

Example 8: Navigating use case levels

The following example shows the relation between two levels of use cases.


Maintain Police Cars
(from Service/Maintenance management)
<<trace>>
<<trace>>
<<trace>>
Service Cars
(from Service/Maintenance management)
Track Police Cars Usage
(from Service/Maintenance management)
Fix Car After Accident
(from Service/Maintenance management)

The *Maintain Police Cars* use case is the higher-level (parent) use case for *Service Cars*, *Track Police Cars Usage* and *Fix Car after Accident* use cases.

Navigating the model is easy, using the two questions discussed earlier – "Why" and "How", e.g. :

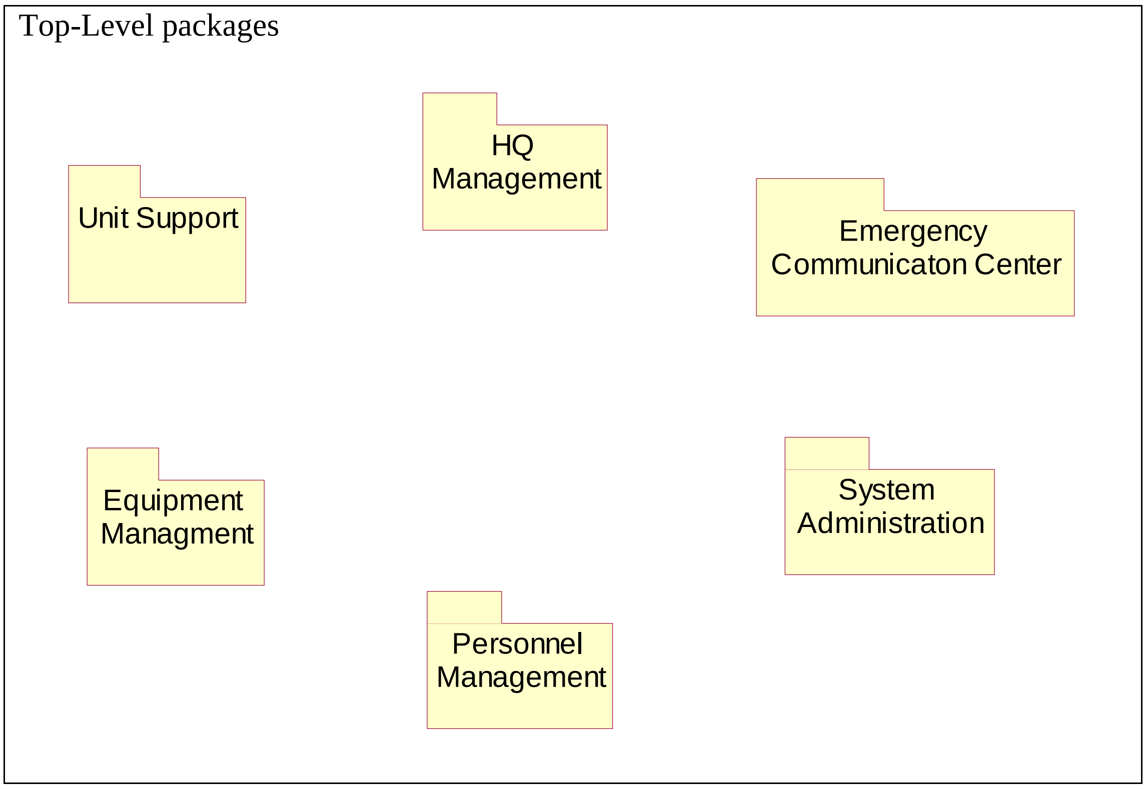
- Why do we track the cars' usage? – In order to maintain the cars
- How do we maintain the cars? – We service them every 10000 km; We fix them after accidents etc.

Additional ways to organize the model are based on categorizing the use cases is by category sets (related group of categories) [Gabb 2001]. Commonly used category sets include importance or priority (more on that in the next section), status (draft, approved, validated etc.), scope (global, local, specific component), stakeholder etc. One important category set is subject – the categories used for this category set are from the problem domain. When the model has a lot of use cases it is also possible to build the subject category as a hierarchy (again to allow easier navigation).

Example 9: Organizing the use case model

The following is an example of the category hierarchy for the Police Force Command & Control system.

The hierarchy can be traversed level by level:

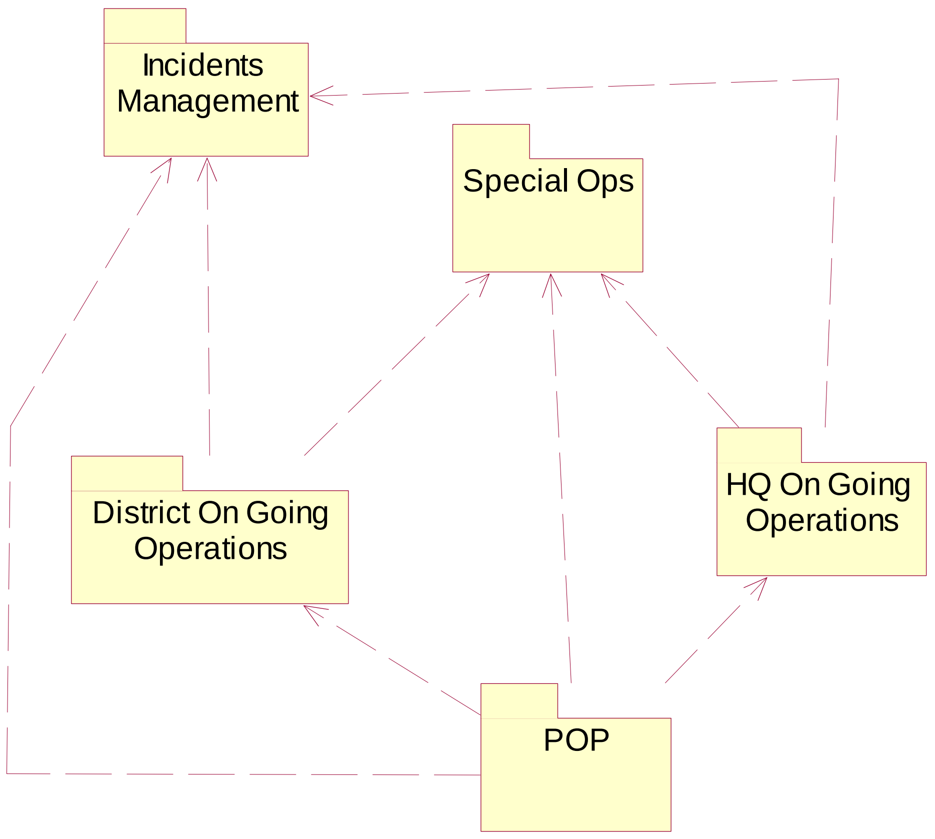


The top level packages cover the 6 major subject that the system has to deal with

- HQ Management – HQ management is the main motivation for building the system. This package include the day-to-day command and control at the district and HQ levels (see below)
- Emergency Communication center – Managing the "911 call center" - includes accepting calls, dispatching units and also the management of the call center personnel
- Unit Support – This package includes the use cases to support the men in the field – navigation, ticketing, connection to the Emergency Comm. Center etc.
- Equipment Management – This is the package that holds the use cases the deal with the logistics side of the operation, taking care of the cars, sensors etc., dispatching technicians to fix faulty equipment, running the spare-parts warehouse etc.

- Personnel Management – This package holds supporting use cases for management of the people of the police force (including for example, allocation to watches etc.)
- System Administration- Includes the use cases for supporting the on-going operation of the system itself (users, maintenance, security etc.).

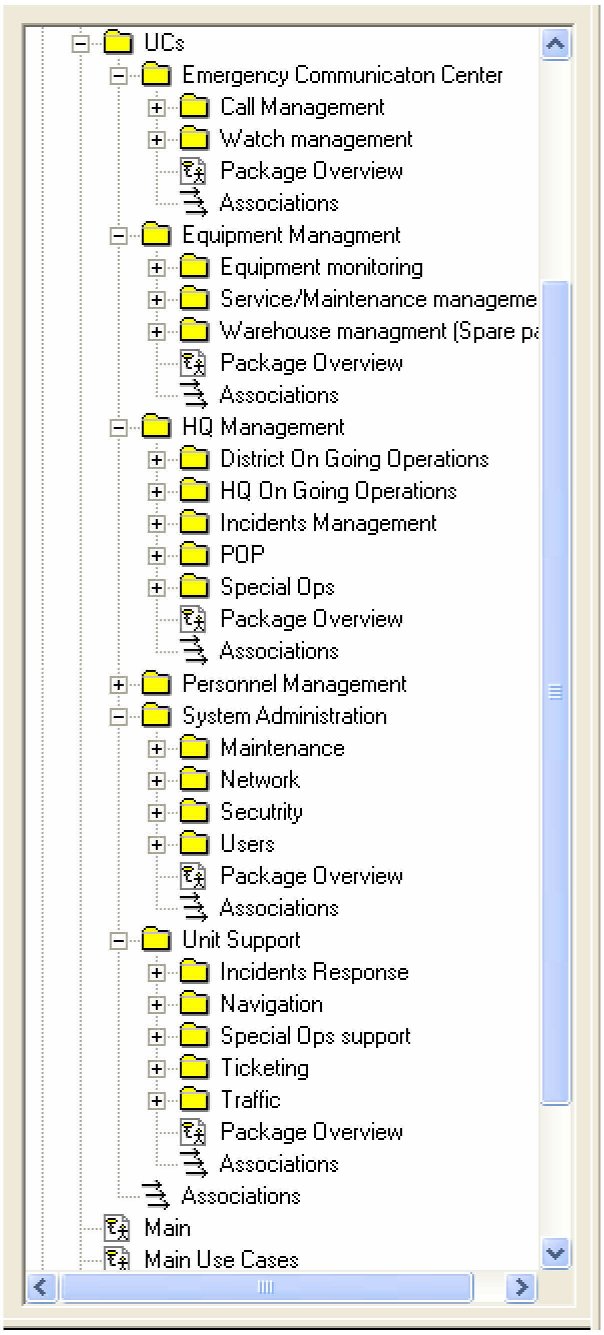
Level 2 (for HQ Management)



The second level use cases (for HQ management packages) is further divided to

- Incidents Management – dealing with incidents that are big enough to bubble to the district level.
- Special Ops – Planning and execution of special operations (planned in advance)
- District On Going Operations – use cases that deal with the day-to-day watch management.
- HQ On Going Operations – same as the above – but on the headquarters level.
- POP – Problem Oriented Policing – use cases that deal with the proactive analysis and response to crimes

In addition to level by level traverse, CASE tools (Rational Rose in this case) also allow browsing the hierarchy as the tree it is.



```
UCs
  Emergency Communicaton Center
    Call Management
    Watch management
    Package Overview
    Associations
  Equipment Managment
    Equipment monitoring
    Service/Maintenance manageme
    Warehouse managment (Spare pa
    Package Overview
    Associations
  HQ Management
    District On Going Operations
    HQ On Going Operations
    Incidents Management
    POP
    Special Ops
    Package Overview
    Associations
  Personnel Management
  System Administration
    Maintenance
    Network
    Secutrity
    Users
    Package Overview
    Associations
  Unit Support
    Incidents Response
    Navigation
    Special Ops support
    Ticketing
    Traffic
    Package Overview
    Associations
  Associations
Main
Main Use Cases
```

When the use case model is large (which is common for a large project), it is beneficial to create views of the use case model from the point of view of the different stakeholders. One such view is the architectural use case view, defined in the RUP [Kruchten 2000], which captures the use cases that are meaningful for determining the architecture. Other options for views include the use case identified per sub-system, per development team, etc. In this respect, it is beneficial to use a requirement management tool that allows filtering and slicing the model to the different views (based on the different categories used).

Step 7: Prioritize use cases

The use cases, being a representation of the requirements, drive the development effort. Modern software projects are built using an iterative process – this is done both to have a better control on the project and its progress and to mitigate risks early [Bittner 2003]. As mentioned earlier, handling all the use cases in a single pass is both counter-productive (overwhelming) and wasteful (requirements change). Prioritizing the use-cases, allows the division of the modeling effort between the different iterations in a way the will contribute, or more precisely, drive the whole development effort.

Note that there is a subtle but important difference between smaller projects and larger ones – whereas in small projects use case prioritization is done to determine what parts of the project will be developed in each iteration, in larger projects the prioritization also determines which use cases will be analyzed in each iteration.

Risk is the key factor for use case prioritization. Three classes of risks are relevant for use-case prioritization: business risks, architectural/technical risks and logistical risks [Bittner 2003].

- Business risks should be mitigated in the initial phase - the main concerns are: Are we building the right product? Is it feasible? What's its cost? Etc. The list of critical (concept level) use cases should be formalized from the use cases identified at this level.
- Architectural/Technical risks are the next level of concern – now the use cases that are significant for the architecture should be found. Additionally issues within the use case that are technically challenging should also be dealt with and analyzed.
- Logistical risks are the 3rd level of concern – now use cases should be described in a manner that will not delay the progress of the deliverables needed (in each iteration).

Example 10: Use cases by risk level

Usually different levels of use cases are identified at each risk level.

Business Risks

Top-level use cases are usually dealt with at this level e.g.

Run Watch, Formalize Problem, Respond to Incident, Handle Emergency Call Handle request for Immediate Assistance.

Risk assessment at this stage reveals, for example, that the "personnel" group of use cases (with use cases like *Recruit Person* and *Retire Officer*) is the least important aspect of the system (per the goals set), and it can either be dropped or postponed to the last stages.

Architectural / Technical Risks

The use cases at this risk level are the ones that should be analyzed first to help propel the development effort.

In the Force Command & Control system use cases that deal with technical and architectural risk can include (few examples):

Get Updated Situation Awareness Picture – for performance requirements on the map module and communications (e.g. how often will the different vehicles report their whereabouts or how much data should the map display at any given time).

Dispatch Unit – for performance and security requirement on the communication system

Scan Problem – for the types of data that has to be captured in the other system use cases to enable this use case.

Logistical Risks

Once the iteration and increments plan has been set use cases are prioritized in a way that will support it – if the first increment includes the emergency communication system then the *Handle Emergency Cal*l and *Handle request for Immediate Assistance* will be decomposed first. Note that use cases like *Respond to Incident* should also be decomposed, at least to some extent (e.g. to the main success scenario level) as they encapsulate important interfaces with the Emergency Communication Center.

The obvious prioritization technique is to handle the use cases by level, first complete the summary level use cases, and then progress to the user goals level etc. The problem is that this approach is too simplistic - While it is true that describing the first couple of levels (at least to some extent) can give a good overview of the problem (help mitigate the business risks), but for the next risk level (mitigate architectural/technical risks) - there are many lower level use cases that should be analyzed – which are significant to the architecture or bear technical risks that should be mitigated early and these use cases, more often than not, are scattered at different levels which makes a level by level approach is wasteful (of time).

Use cases are prioritized in relation to the other use cases, this means that at a certain stage, too many use-cases (to fit in the timeframe allocated within the current iteration) will be chosen. There are several mathematical models that can be used to resolve such cases [Moisiadis 1998] including, for example, Quality Functional Deployment (QFD), Analytical Hierarchy Process (AHP), Fuzzy Multi-Criteria Decision making, etc.

It should be noted that priority should be assigned not just at the use-case level, but also at the scenario level [Armour 2001] to allow a more fine-grained control over the development progress.

It is important to remember that use case prioritization can only begin after an ample number of use cases has been discovers/described to obtain a meaningful understanding of the system and its risks.

Step 8: Describing the use cases

Up to this step – the use case model is rather thin on details about each use case. The model has a set of identified use cases (each with a short description and possibly with pre and post conditions); there are probably some initial diagrams depicting the use cases relations and hopefully some categories and priorities for the different use cases. It is in this step, that the use cases at the current priority level (per iteration) should be elaborated.

It is important to use a standard template for describing the use-cases. Using templates have several advantages, including making it easier for the team to know what information is expected in each use case description; making the writing effort more precise (providing the purpose of each use-case component is clear and defined) and allowing easier understanding of the use cases (figuring out the structure of one – allows the reader to know what to expect from the rest) [Adolph 2003].

There are many use case templates available, however many of them [Armour 2001, Kruchten 2000, Kulak 2000, Cockburn 2001] share basically the same structure (though, sometimes using slightly different names). A template should include the basic fields already mentioned in the "find use case" step (name, scope, actors, preconditions, success guarantee and trigger) and, in addition, the following fields:

- Main Success scenario – the interactions between the system and the actors to accomplish the goal of the use case. This is the "sunny day scenario", where no errors occur and the most common actions are performed.
- Variations – less common interaction paths (relative to the main success scenario)
- Exceptions – the interactions with the system when errors occur
- Assumptions – any assumption made (can be complimentary for pre-conditions as things that the system cannot guarantee)

Additional useful fields include:

- Status [Kulak 2000] – the current state of the use case (first draft, validated, filled, finished etc.)
- Priority [Armour 2001, Cockburn 2001] – The development priority of the use case (see the discussion on prioritizing use cases)
- Stakeholders and concerns [Cockburn 2001] - stakeholders (that are not necessarily actors in the use case) and their interests (that must be protected within the use-case)
- Issues [Armour 2001] – any open issues in regard to the use case (this should be empty when the use case is declared finished)
- Non-behavioral requirements [Kruchten 2000,Armour 2001] – any non-functional requirements that are related to the use-case (performance, security, safety etc.)

- Extension Points [Kruchten 2000, Kulak 2000] – steps in the use case where the extending use cases diverge (more on "extend" relation later).

Lastly, in regard to misuse cases, specific additions can be considered [Sindre 2001]:

- Capture point –options where the misuse case can be prevented or detected.
- Worst case threat (instead of success guarantee) – what happened if the misuse case successes
- Detection guarantee - describes the outcome when a prevention scenario is followed
- Prevention guarantee - described the outcome if a detection scenario is followed

Usually it is best to use a predefined standard template (such as the ones mentioned above) or, alternatively, to mix-and-match from the various sources to build a template suitable for the specific project's needs.

Having a template takes care of the formal side of describing the use cases. There are, however, additional aspects that must be considered. The first of which, is that the use case text should focus on the use case goal (the one that originated the use case in the first place).Use cases that try to cover a lot of ground (try to act as a "swiss-army knife") have a negative effect on managing complexity. Also use cases that get too large can obscure and bury both the purpose of the use case and the stakeholders needs [Adolph 2003].

Another important issue is to keep the use cases - technology neutral [Adolph 2003]. Technological details clutter the use cases with information that, more likely than not, is obscure and cryptic to customers and will thus hinder communication – which is one of the reasons for writing use cases to begin with. Also since technology is volatile and may change during the project course, detailing technology increase the risk of rework to maintain the use-cases up-to-date.

There are many additional points and techniques in regard to writing use-cases that are not in the scope of this paper (i.e. they pertain to use-case in general). In this respect, it can be beneficial to read books, such as Alistair Cockburn's "writing effective use cases" [Cockburn 2001], that deal specifically with the single use case.

In regard to iterative development, it is important to note that the word *iterative* here is used in two levels - not all the use cases will be described in a specific iteration, and also that not all the use case is necessarily described in any single iteration. The risks that drive the prioritization of the use-case set also drive the elaboration of the single use case.

In addition to the textual representation of the use case, it is sometimes beneficial to diagram the use case course, for example when the sequence of events within the use case (with all the exceptions and variations) is complex and difficult to follow. It is recommended, in these cases, to use UML's activity diagrams [Armour 2001] to visualize the scenarios. Activity diagrams are suitable for the job since they allow describing parallelism, iterations and conditional logic [Fowler 2000] (the recently announced UML2.0 also allows describing these behaviors using Sequence diagram).

Example 11: Using Activity diagrams

The following example shows a use case description followed by its representation as an activity diagram.

Use Case: Handle Emergency Call

ID: UC24

Scope: The Operator accepts an incoming call, enters the incident information and dispatch a unit to the location of the incident

Stakeholders and Concerns:

- Victim - wants the police to arrive as soon as possible
- Beat Team – don't want to be dispatched to handle false incidents.

Primary Actor: Emergency Center Operator

Preconditions: Operator logged in.

Success Guarantees:

- The Call has been recorded
- A unit has been dispatch to investigate the incident
- The incident details are saved in the system

Trigger: A Citizen's incoming call has been directed by the Call Center system to an Operator.

Main Success Scenario:

1. The system begins recording the call.
2. The system traces the caller address.
3. The Operator takes the incidents location
4. The system calculates available police units.
5. The Operator takes the incidents detail
6. The system presents a list of available teams and their distance from the incidents estimated location.
7. The Operator chooses a unit to handle the incident
8. The system dispatches the incident details to the chosen team.
9. The Operator takes the caller details
10. The system saves the incidents details including call statistics
11. The system ends recording.

Variations:

1. step 2 - when the caller uses a mobile phone
    a. Locate the callers current location
2. step 2 - when the caller is on the black list (known to call for no reason)
    a. The Operator is presented with additional questions to ask the caller
    b. The system marks the incident as low-priority on count of possible false alarm.
3. step 7 - when the incident does not require police intervention.
    a. The Operator closes the incident
    b. The system saves the termination reasons and continues from step 10
4. step 7 - if the incident requires a fire truck/ambulance
    a. The Operator chooses which authority to notify (fire / ambulance etc)
    b. The system dispatches the incident details to the appropriate authority's system

Extension Points:

1. step 3 – Instantiate use case - handle an emergency call for a suspected reported incident

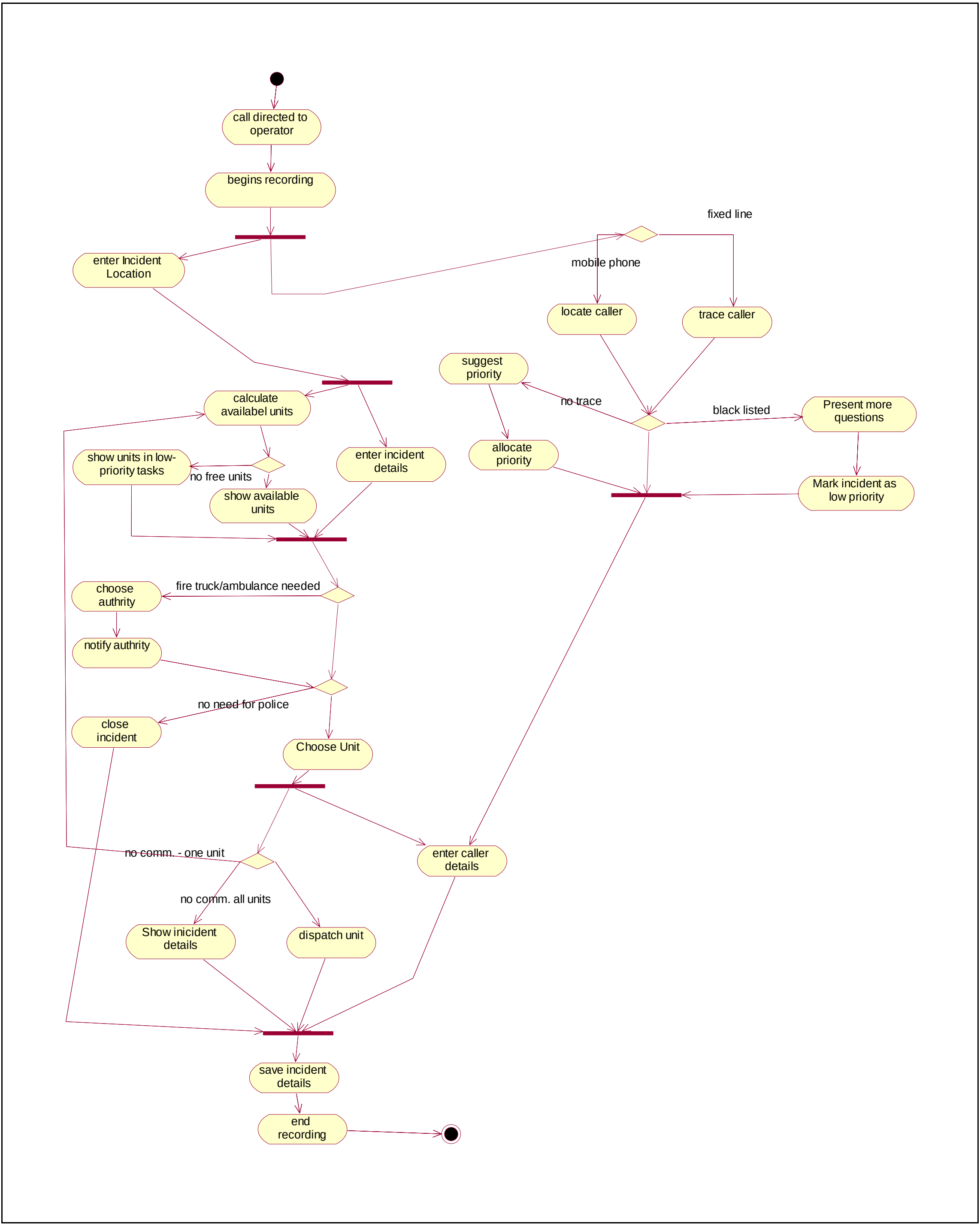
Exceptions:

1. step 2 - when the call cannot be traced
    a. The system suggests lowering the priority of the call on the count of an unknown caller
    b. The operator decides what priority to allocate for the incident.
2. step 6 – when there is no available free force
    a. The system presents the operator with low-priority incidents (along with the reason for low-priority
3. step 8 – communication problem with the unit dispatched
    a. The system performs step 6 and 7 again.
4. step 8 – communication problem with all the units.
    a. The system presents the operator the incidents details to allow dispatching by radio/mobile phone.

Non Behavioral Requirements:

- The system should present as few screen as possible to the operator
- Locating a free unit should take less than 30seconds
- Communications to and from the unit should be secure (encrypted) to prevent eavesdropping by offenders/media

Activity diagram for the "Handle Emergency Call" use case :



An additional type of diagram that can be used is sequence diagrams. However, using Sequence diagrams is usually more beneficial during the use case analysis phase (not in the scope of this paper) since sequence diagrams can show only a single use case thread (a scenario instance). During the analysis phase, this can be used to better understand the components that take part in the use case.

Step 9: Refactor the model

Refactoring is a change made to the internal structure of a component to make it easier to understand and cheaper to modify – without changing the observable behavior of that component [Fowler 1999]. Originally defined for coding (C++), the principle is applicable to other areas of the software world – use case modeling included.

Refactoring use cases can mean several things: identifying common behavior; modeling use case relations, extracting extensions and alternate scenarios to separate use cases, cleaning up the model etc. The next paragraphs will detail with the most common of them.

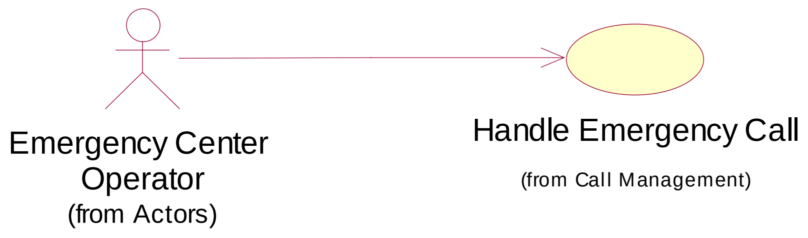
The first type of refactoring methods relate to distributing behavior [Rui 2003]. When several use cases share a common behavior, or more precisely share a course of action [Adolph 2003]. It is recommended to extract the common behavior to a new use case and use the *include* relation to relate it to the parents. Sometimes the common behavior can be considered as a parent use case of the two (or more) use cases the refactoring, in such cases, is similar but instead a parent use case is created and the scenario is moved to the parent and then related the child use cases [Rui 2003].

The *include* relation is sometimes used to relate more detailed versions of other use cases. Some practitioners recommend using *trace* relation for this cases [Adolph 2003] and keep the *include* relation only for common sub behavior.

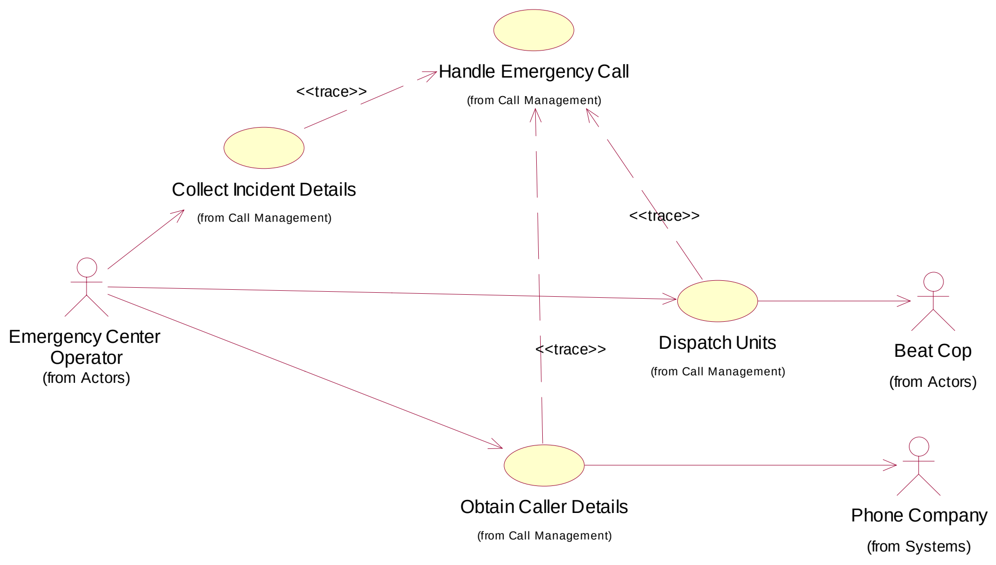
Example 12: Use case relationships – include vs. trace

The example below demonstrates the difference between the *trace* and *include* relations.

At one level we have the Emergency Center Operator, her job is to handle rmergency calls:

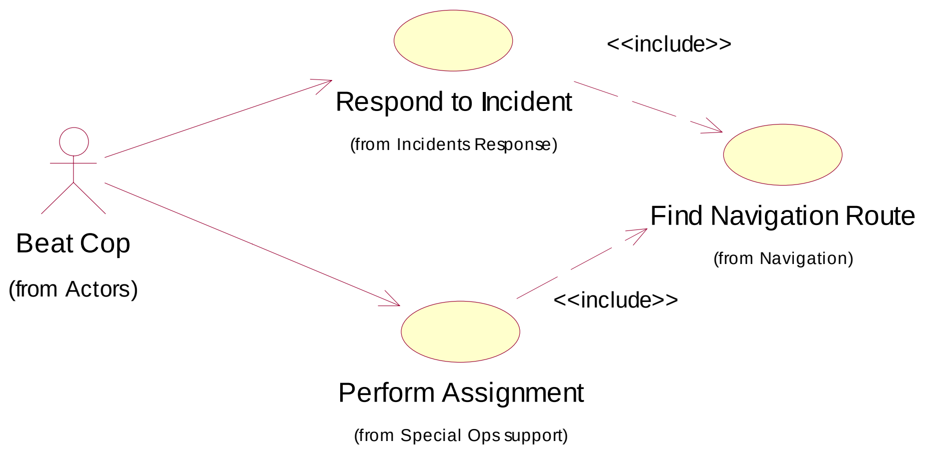


The *Handle Emergency Call* use case can be decomposed into three sub use cases, each of these use cases can be considered as a step in the parent use case.



Note that the diagram doesn't say anything about the order of the sub-use cases, also additional actions can occur when handling an emergency call that are not large enough to warrant a use case by them selves (these actions are detailed in the text of *Handle Emergency Call*).

The *include* relation, however is used to relate common sub-behavior of other use cases:



Both responding to an incident and performing an assignment (as part of a special operation) need to use *Find Navigation Route* – to allow the cop to arrive at the scene as fast as possible (in the *Response to Incident*) or on the designated time (on *Perform Assignment*).

Note that at the solution level the software for *Perform Assignment* and *Respond to Incident* might turn out to be the same – however from the requirements point of view each use case serves a different goal.

Another class of refactoring type is moving elements of use cases [Rui 2003]. When an alternative path affects several steps, it can cause the reader to get confused and loose track between the primary and the alternative paths [Adolph 2003]. It is recommended in these cases to move the alternative path to a use case of its own and relate the two use cases using the *extend* relation. The *extend* relation can also be used when an alternative is too long or complex and thus dominates the use case. Moving this kind of alternatives to their own use case will prevent them from obscuring the real use case.

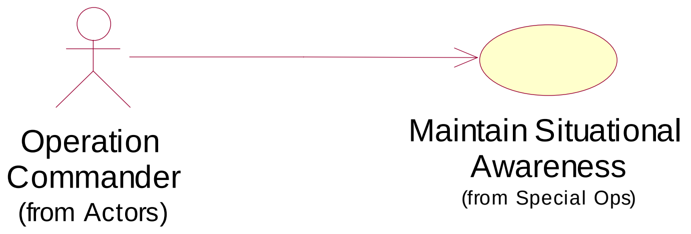
Example 13: Use case relations – using extend

The following example shows the use of *extend* relation for child use cases that are a specialization of the parent use case.

Situation awareness is an intermediate state in the decision making process where the commander has to have [Endsley 1995]:

- Perception of the current environment and how it came about
- Understanding of possible futures

This is, naturally an essential part of carrying out a special operation – thus the following use case was identified.



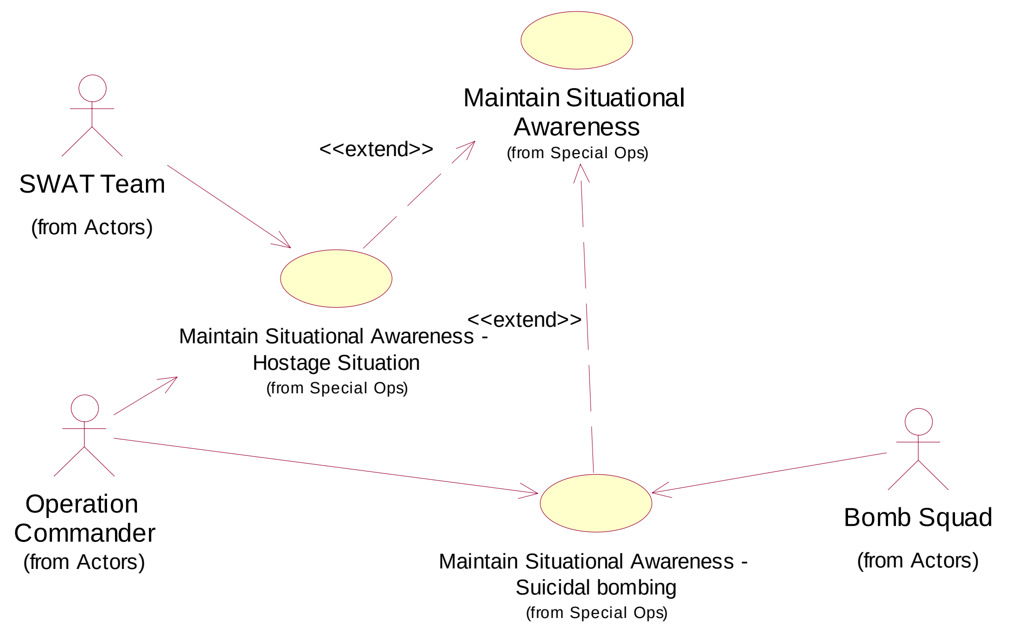
Achieving situational awareness is not something the system does, it is however, something the system can assist gaining - by collecting and displaying all the relevant information regarding the situation (position of forces, estimated position of offenders, operation goal) , supporting what-if analysis etc.

When the types of special operations that exist are considered, several of those can be identified– two examples are handling a suicidal bomber and handling a hostage situation.

While the basic lines of these processes are the same as the general use case defined earlier – there are special steps that each of these use cases needs, for example:

- Hostage situation – it would be beneficial if the system will display in addition to the above mentioned data, additional information on the building where the hostages are held, number of employees, 3D model etc. Also, the operation will be carried out by the SWAT team and they have different characteristics compared with other forces.
- Suicidal Bomber – For this use case we need interaction with the bomb squad, also there are special needs in regards to road blocks as well as crowd control and evacuation plans (assisting ambulances by clearing traffic etc.).

Thus the two use cases (and any other special op. for that matter) will have an extend relation to the genetic use case of Maintaining Situational Awareness as can be seen below:



Another example of moving use case elements is merging related tiny use case fragments into a single use case that relate to the same goal [Adolph 2003]. Tiny use cases usually do not hold user goals and don't help understanding the system.

Merging use cases is also related to the refactoring type of deleting use case elements [Rui 2003]. It is possible that use cases that were identified in earlier stages, do not contribute to the overall understanding of the system, or even worse, distract and cause misunderstandings [Adolph 2003]. Additionally, use case that seems meaningful at first glance, can be left unreferenced as the analysis progresses. A good option is to remove these use cases from the model, the same way dead code would be done with, in software programs.

Additional refactoring type is changing use case elements [Rui 2003] –including changing use case names to better reflect their meaning, updating actors list, refining preconditions etc.

A note about use case relations - UML also defined a 3rd use case relation (in addition to *include* and *extend*) [Fowler 2000] called *generalize*. This relation should be used when a use case is similar to another use case but does something in a different way (different precondition, path etc.) – however, it is not recommended for use as its meaning is hard to grasp for non-programmers.

Supporting Steps

Step 10: Verify & Validate (V&V) the model

The term Verification ("Are we building the product right?") and Validation ("Are we building the right product") is used so much to the point of almost being a cliché – nevertheless – verifying and validating the model is an extremely important step.

The following table, taken from [Anda 2002] ,sums up the different defects that can occur while modeling use cases.

UC Element / Problem type	**Actors**	**Use cases**	**Flow of events**	**Variations**	**Relation between use cases**	**Trigger, pre- and post-conditions**
Omissions	Human users or external entities that will interact with the system are not identified	Required functionality is not described in use cases. Actors have goals that do not have corresponding use cases	Input or output for use cases is not described. Events that are necessary for understanding the use cases are missing	Variations that may occur when attempting to achieve the goal of a use case are not specified	Common functionality is not separated out in included use cases	Trigger, pre- or post-conditions have been omitted
Incorrect facts	Incorrect description of actors or wrong connection between actor and use case	Incorrect description of a use case	Incorrect description of one or several events	Incorrect description of a variation	Not applicable	Incorrect assumptions or results have led to incorrect pre- or post-conditions
Inconsistencies	Description of actor is inconsistent with its behavior in use cases	Description is inconsistent with reaching the goal of the use case	Events that are inconsistent with reaching the goal of the use case they are part of	Variations which are inconsistent with the goal of the use case.	Inconsistencies between diagram and descriptions, inconsistent terminology, inconsistencies between use cases, or different level of granularity	Pre- or post-conditions are inconsistent with goal or flow of events
Ambiguities	Too broadly defined actors or ambiguous description	Name of use case does not reflect the goal of the use case	Ambiguous description of events, perhaps because of	Ambiguous description of what leads to a particular	Not applicable	Ambiguous description of trigger, pre- or post-condition

	of actor		too little detail	variation		
Extraneous information	Actors that do not derive value from/provide value to the system	Use cases with functionality outside the scope of the system or use cases that duplicate functionality	Superfluous steps or too much detail in steps	Variations that are outside the scope of the system	Not applicable	Superfluous trigger, pre- or post-conditions
Consequences	Expected functionality is unavailable for some users or interface to other systems are missing	Expected functionality is unavailable	Too many or wrong constraints on the design or the goal is not reached for the actor	Wrong delimitation of functionality	Misunderstandings between different stake-holders, inefficient design and code	Difficult to test the system and bad navigability for users between different use cases

There are basically four approaches to verify & validate a model [Anda 2002, Hansen 2002], and a mixing and matching several of them, is usually needed for optimal results:

- Inspections (verify & validate) – the act where an individual or a team looks at the use-cases according to pre-defined criteria to verify their adherence to standards and specifications. Appendix B holds a set of questions that can serve as a check list for use case model inspections (and reviews).
- Reviews (verify & validate) – involve multiple readers examining the different use case artifacts (text, diagrams). Reviews should involve customer representatives and other stakeholders (as already mentioned in organize the team step – it is beneficial to hold the reviews in two tiers – small internal team and a second review with the complete group [Adolph 2003]).
- Walkthroughs (Validate) - a form of review where a use case or a business scenario (comprised of several use-cases interacting) is actively presented (usually by the author), and possibly role-played in-order to examine the flow of events.

- Prototyping (Validate) – This is based on turning rapid prototype (most likely screen mockups) to demonstrate to stakeholders (esp. the customer) the behavior depicted in the use-case. The advantage of this approach is the visibility of the understanding captured by the use case. The disadvantage, of course, is the added costs.

No matter which of these approaches is chosen, it is important to remember that proper verification & validation help reduce the risks of the problems described above and improve the quality of the project, not to mention the customer satisfaction.

Step 11: Add future requirements

Adding future requirements is not about requirement management (guarding against feature creep etc.), rather it is about features that are identified during the requirement analysis, which are not planned to be developed but it is anticipated that they will be developed as future enhancements. These requirements are also known as "provision for" requirements. Describing such requirements is beneficial, especially when it is identified that these future requirements influence the overall architecture of the solution, so that addressing such requirements now will indeed allow for them to be integrated more easily.

The UML construct that deals with such changes is called a *change case* – These are basically regular use cases with a couple of differences [Ecklund 1996]:

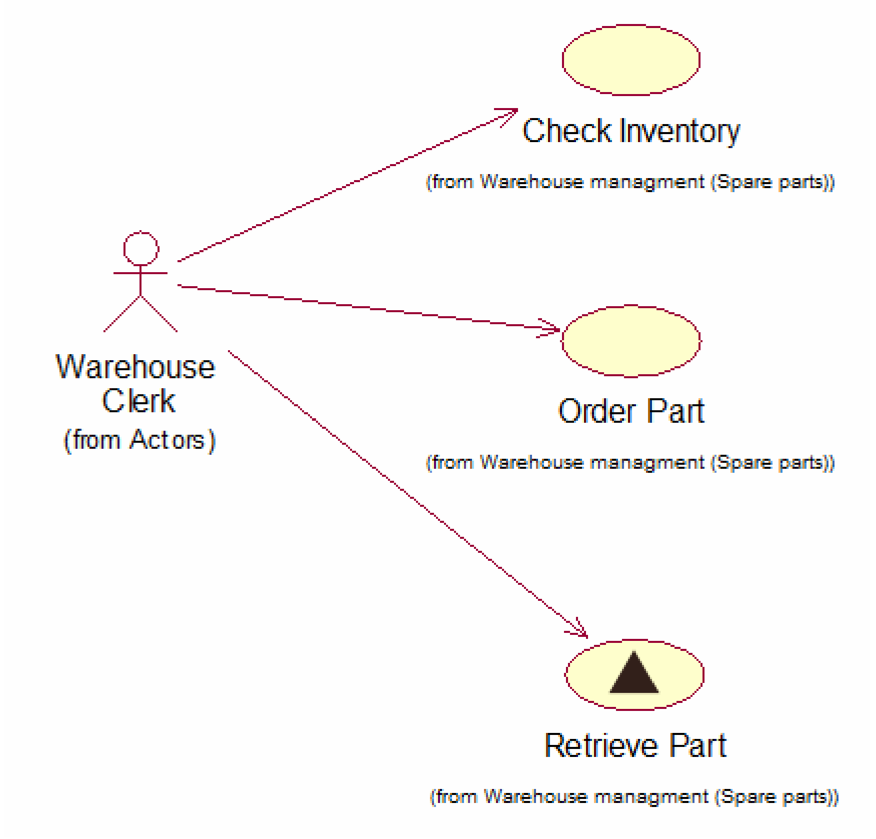
- (by definition) they talk about process/goals that will not be implemented in the current system
- Each change case holds traceability information to identify use cases that will be affected by the change

The main benefits of using change cases are:

- The design, and more importantly the architecture, can take into account the requirements and prepare for the change.
- Impact analysis that is already pre-made when the change materializes. – The impact can be assesed by dividing the number of use cases affected by the change to the total number of use cases in the same level as the ones affected by the change [Ecklund 1996].

Example 14 Change cases

The following example shows a future enhancement to the Police Force Command & Control system, along with a couple of other use cases that exist within the current scope (which can be affected by the change case).


Check Inventory
(from Warehouse managment (Spare parts))
Warehouse Clerk
(from Actors)
Order Part
(from Warehouse managment (Spare parts))
Retrieve Part
(from Warehouse managment (Spare parts))

Warehouse management is part of the scope of the current system. Two of the use cases identified to support this capability include *Check Inventory* and *Order Part*.

- *Check Inventory* – allows the warehouse clerk to check her computer and see if any needed part in stock. If it is the clerk will have to go and retrieve it – otherwise she will have to order a new part.
- *Order Part* – This use case takes care of everything needed to get a new part for the system – from filling in the purchase order to adding it to the stock

The *Retrieve Part* use case is a future enhancement – under the current solution it is the clerk's responsibility to know where she placed the part, and then manually go and retrieve it and decrease the number in the stock. The future enhancement is to add a robot that will automatically store and retrieve parts. It is important to detail such requirements today as adding such a robot, affects the numbering scheme for the different spare parts. It can also have effect on the technology that will be used to catalogue the parts (bar codes vs. plain text labels) etc. Capturing this change case now, allows planning the system today in a way that will allow the system to grow in that direction with the minimal changes.

It is impossible (and not practical) to predict all the changes that will occur in a project, but predicting extensions that are likely to occur is not that complicated [Armour 2001]. Working on future requirements means that the team is not working on the requirements for the current project – thus it is important to limit the number of change cases in the model to a selected few and not derailing the main effort of developing the use case model for the part of the system that will be developed.

End Game

Step 12: Knowing when to stop

It is very important to devise quitting or stopping criteria for the modeling effort – especially considering that use cases can always be decomposed to smaller and smaller fractions and tasks almost ad infinitum. Deciding when to stop is a delicate balance of the risk of incomplete requirements versus the risk of delaying the project (or increasing its running costs needlessly).

The following questions can be used to base the stopping criteria [Adolph 2003]

- Have all the actors and goals been covered?
- Has the customer acknowledged that the model is complete and that the use cases readable and correct?
- Can the designers implement the use case?
- In case of a system of systems: Has the requirements for each sub-system have been identified within each use case?

Deciding when to stop is not an irreversible decision - should the need arise, the use cases can be revisited and detailed further. However, re-opening the use case model, once it has been closed, should probably be approved by a change committee (and/or other requirement change process) – to make sure there is reason enough for the step.

In addition to the final stopping decision, developing iteratively means that there will be several other, more minor, decision points (per iteration) - on when are the use cases for the iteration complete. The questions that should be answered (to support these decisions) are simpler:

- Have all the use cases that were prioritized for this iteration been detailed?
- Has the level of detail, agreed upon for the iteration been reached for each of these use cases (and for any child use case that was spawned by analyzing them)?

Summary

Use case modeling is a powerful technique for capturing system requirements, however, like any other technique; there are several challenges in applying and scaling a theoretically promising concept to real-life projects. A lot of the books and resources available are lacking in this sense, as the examples and methodology depicted in them is only suitable for smaller projects and doesn't scale well.

The methodology described in this paper, is based both on industry best practices and on hard earned experience gained from participating in several large projects in the defense, telecom and business intelligence industries.

I have covered some of the major challenges that use case modeling poses to large projects, and demonstrated and detailed the steps needed in order to mitigate these risks and thus help achieve a use case model that will be usable by both the project team, the customer and other stakeholders.

The methodology, as it is described in this paper, only deals with the development of the use case model, there are several other aspects of the development life-cycle that can have a significant effect on the success of the modeling effort.

The most important areas that can make or break the use case modeling efforts are:

- Requirements management - managing the changes in requirements, guarding against unmanaged feature creep, traceability to the RFP (if it exists) etc.
- Configuration management - related to the former issue, in regards to versioning of the use cases. This is also important in regulating and coordinating the team work of developing the model.
- Project Management – Setting the priorities right, dividing the iterations correctly, managing the availability of suitable personnel for the job etc.
- Maintaining the teams focus and drive – Related to project management, this means managing the use case development team, not letting them get lost in the details – it is also related to the "knowing when to stop" step described earlier.

Lastly, In large organizations the methodology will need tailoring both to match the organization's culture and the specific project's needs. I do believe, however, that applying the methodology (or the key concepts of the methodology) described in this paper, can help increase the chances of a successful modeling effort for large and small projects alike.

Appendix A: Police Force Command & Control (C&C)

As mentioned in the preface, the examples used in the paper to demonstrate key issues are in fact small fractions of a large problem. The following section describes the overall framework or the problem statement within which all the examples exist.

The project at hand is a comprehensive Force Command & Control system for police headquarters.

Background:

There are six operational strategies (five are core and one auxiliary) that police forces use for achieving the policing goals [Scott 2000]:

- Preventive Patrol - the idea is to have presence - i.e. police officers in uniform patrolling the streets. The logic behind this is twofold. First the present of police officers is expected to deter citizens from committing offenses and enhance the sense of security for the law abiding citizens. Second, the presence of police officers is expected to increase the probability that they will interrupt offenses in progress.
- Routine Incident Response – the regular day-to-day work, of responding to "calls for service", include for example, restoring order, document complaints, etc.
- Emergency Response - This strategy objective it to save lives, minimize injuries, restore basic level of order etc. It encompasses crimes in progress, officers' requests for immediate assistance, traffic accidents with injuries, natural disasters etc.
- Criminal Investigation - once the police determines that a crime has been committed (usually triggered by Routine or Emergency Response) – this strategy provides the framework for investigations . The unit of work is the case and processes to collect enough evidence to solve the "cases" (usually by apprehending a suspects and bringing them to justice).
- Problem Solving – is a methodology for proactively dealing with the policing problems. It involves a process for problem identification, analysis, response and evaluation.
- Support services – This includes activities such as teaching crime prevention techniques, operating youth activity programs, providing copies of police work

and other general services that are not related to the other 5 operational strategies.

Problem Statement:

The solution should provide services and support the more proactive aspect of the policing work- namely: Preventive Patrols, Emergency Response and Problem Solving.

The solution has to cover three main areas:

A. on-going operations

   a. The system has to support an "Emergency Communications Center", manage a dispatch office of several operators, track forces allocation to incidents/calls, and allow allocation of forces (i.e. dispatch units) to new incidents etc. The system should also collect "classic" call center statistics such as number of calls, response time of operators, complaints etc.

   b. Allow planning and carrying out of special operations (e.g. a large drug busts, Terrorist hunts etc.) – The system should support force allocation to missions, planning of routes etc., continuous near real-time tracking of participating forces, progress and completion reports.

   c. The system should supply a "situation awareness" picture so that the police HQ will be able to monitor the overall situation of all police cars and other police forces in each of the districts. The same capability should be available at the district level.

   d. The system should support incident pattern analysis i.e. help police officers identify and analyze problems, as well as assess the success of responses. Officers should be provided with crime and incident data relating to their beats on a monthly basis. The system should also interact with external systems (such as an Area’s Intelligence Unit) import and export data. Incidents should also be mapped to street / neighborhood level to enable spatial analysis of incident patterns.

B. Aids for police personnel on the field – It is required that the police cars / policemen will be equipped with wireless terminal (e.g. PDA) that will have on-line connection to the HQ, District and Emergency Communications centers. These terminals should support the following functionality :

   a. Navigation maps (GPS based) – including current-location reporting to HQ/district.
   b. List of duties (running list of in progress (emergency) calls/operations allocated to the force)
   c. Link to the police offenders and criminals records (a legacy system already in-service) to allow a policeman to check data on suspects etc.
   d. Link to the car and drivers registry (an external database run by the ministry of transport) – to allow policemen to key SSN (Social Security Number) or VIN (Vehicle Identification Number) and retrieve license and insurance status for both the driver and car.
   e. Ticketing and reporting system to allow policemen to issue tickets and file event reports.

C. Logistics Management – the system should manage

   a. Sensor management – collecting data on-line from sensors (radars, camera etc.) regarding their operational status and mange the technicians that service the sensors etc.
   b. Police Cars management – track usage of police cars and mange the servicing of cars, trucks etc. (when to send a car to the garage etc.)
   c. Personnel management – mange which policemen are assigned to which car/unit , which is on leave etc.

In addition to the above mentioned "business" aspects of the system, it should also support system administration functionality (adding/ removing users, issue passwords, network monitoring and management). The system should also have a rigorous and robust security system (to prevent unauthorized retrieval / alteration of data).

Appendix B: Use Case inspection questions

This section lists several questions, from various sources [Carr 2000, Anda 2002, Armour 2001] as well as personal experience, that can serve as a check list for use case model verification and validation.

Actors

- Are there any actors that are not defined in the use case model, that is, will the system communicate with any other systems, hardware or human users that have not been described?
- Are there any superfluous actors in the use case model, that is, human users or other systems that will not provide input to or receive output from the system?
- Are all the actors abstractions of specific roles?
- Are all the actors clearly described, and do you agree with the descriptions?
- Is it clear which actors are involved in which use cases, and can this be clearly seen from the use case diagram and textual descriptions?
- Are all the actors connected to the right use cases?

The use cases

- Does the use case make sense?
- For each iteration: Are all the use cases described at the same level of detail?
- Is there any missing functionality that is, do the actors have goals that must be fulfilled, but that have not been described in use cases?
- Are there any superfluous use cases, that is, use cases that are outside the boundary of the system, do not lead to the fulfillment of a goal for an actor or duplicate functionality described in other use cases?
- Do all the use cases lead to the fulfillment of exactly one goal for an actor, and is it clear from the use case name what is the goal?
- Are the descriptions of how the actor interacts with the system in the use cases consistent with the description of the actor?
- Are the actors external to the use case boundary?

- Is it clear from the descriptions of the use cases how the goals are reached and do you agree with the descriptions?
- When there is an RFP document: Is there bi-directional tractability between the use cases and the originating requirements?
- Are the use cases testable?
- Are all the use cases described according to the predefined template?
- Do all the use case names follow the naming convention (most likely verb-noun)?

The scenarios

- Is the start of each use case unambiguous?
- Does an action by an actor start each use case?
- Is expected input and output correctly defined in each use case; is the output from the system defined for every input from the actor, both for normal flow of events and variations?
- Does each event in the normal flow of events relate to the goal of its use case?
- Is the flow of events described with concrete terms and measurable concepts and is it described at a suitable level of detail without details that restrict the user interface or the design of the system?
- Are there any variants to the normal flow of events that have not been identified in the use cases, that is, are there any missing variations? ("happy days scenarios", exceptions, variation, "soup-opera scenarios")
- Are the triggers, starting conditions, for each use case described at the correct level of detail?
- Are the preconditions and guarantees correctly described for all use cases, that is, are they described with the correct level of detail, do the preconditions and guarantees match for each of the use cases and are they testable?
- Does the behavior of a use case conflict with the behavior of other use cases?
- Is the number of steps in the complex scenarios excessive (12 to 15 is getting borderline)?

The use case diagrams

- Does each use case have a representation in at least one diagram?
- Do the use case diagram and the textual descriptions match?
- Are all use case diagrams drawn using the same (preferably the UML's) diagramming notation?
- Is each actor represented in the use case diagrams in which it is involved?
- Should similar use case diagrams be combined (using *extend* and *uses* relations)?
- Has the include-relation been used to factor out common behavior?
- Are the diagrams readable (not too many relations, levels etc. in any single diagram)?

The use case organization and prioritization

- Are all the use cases organized in an appropriate manner (e.g. by functional area, by dependency, by actor etc)?
- Are all the use cases within a package consistent with the theme of the package?
- Is the priority mechanism documented?
- Are the use cases prioritized correctly?

Bibliography

[Adolph 2003]	W. S. Adolph and P. Bramble, *Patterns for effective use cases*. Boston ; London: Addison-Wesley, 2003.
[Alexander 2002]	I. Alexander, "Initial Industrial Experience of Misuse Cases in Trade-off Analysis", P*roceedings of IEEE Joint International Requirements Engineering Conference*, 9-13 September 2002, Essen, pp 61-68
[Anda 2002]	B. Anda, , D. I. K. Sjøberg "Towards an inspection technique for use case models", *Proceedings of the 14th international conference on Software engineering and knowledge engineering* , 2002. pp. 127-134.
[Armour 2001]	F. Armour and G. Miller, *Advanced use case modeling : software systems*. Boston ; London: Addison-Wesley, 2001.
[Bittner 2003]	K. Bittner and I. Spence "Managing Iterative Software Development with Use Cases," *The Rational Edge*, 2003.
[Booch 1999]	G. Booch, I. Jacobson, and J. Rumbaugh, *Unified Modeling Language Users Guide*: Addison Wesley Longman, Inc, 1999.
[Brooks 1995]	F. P. Brooks, *The mythical man-month : essays on software engineering*, Anniversary ed. Reading, Mass.: Addison-Wesley Pub. Co., 1995.
[Carr 2000]	J. T, Carr III, O. Balci, "Verification and Validation of Object-Oriented Artifacts Throughout the Simulation Model Development Life Cycle ", *Proceedings of the 2000 Winter Simulation Conference*, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds. , 2000.
[Chicago 2003]	Chicago police dept. web site,http://egov.cityofchicago.org/city/webportal/portalEntityHomeAction.do?entityName=Police&entityNameEnumValue=33, 2003.
[Cockburn 2001]	A. Cockburn, *Writing effective use cases*. Boston ; London: Addison-Wesley, 2001.
[Crain 2002]	A. Crain, "Dear Dr. Use Case: Is the Clock an Actor," *The Rational Edge*, 2002.
[Ecklund 1996]	E. F. Ecklund, Jr., L.M.L. Delcambre, M. J. Freiling, "Change cases: use cases that identify future requirements", in: *Proceedings OOPSLA'96*, 1996, pp. 342-358.
[Endsley 1995]	M. Endsley, “Towards a Theory of Situation Awareness in Dynamic Systems,”," *Human Factors*,, 37:1, 1995, pp. 32-64.
[Firesmith 1996]	D. G. Firesmith, "“Use Cases: The Pros and Cons,”," in *Wisdom of the Gurus: A Vision for Object Technology*,, Charles F. Bowman, Ed. New York: SIGS Books Inc., 1996, pp. 171-180.
[Fowler 1999]	M. Fowler, *Refactoring: Improving the Design of Existing Code,* ; Harlow, England: Addison-Wesley, 1999.
[Fowler 2000]	M. Fowler and K. Scott, *UML distilled : a brief guide to the standard object modeling language*, 2nd ed. Reading, Mass. ; Harlow, England: Addison-Wesley, 2000.

[Gabb 2001]	A. Gabb et al "Requirements Categorization", *http://www.incose.org/rwg/01_req_categories/categories.html,* INCOSE requirements working group , 2001.
[Gottesdiener 2003]	E. Gottesdiener "Use Cases: Best Practices", *http://www.rational.com/media/whitepapers/usecase_bp.pdf,* Rational Software , 2003.
[Ham 1998]	G. A. Ham, "Four Roads to Use Case Discovery," *CrossTalk,* vol. 11, 1998.
[Hansen 2002]	T Hansen, G. Miller "A Framework for Prioritizing Use Cases Definition and Verification of Requirements Through Use Case Analysis and Early Prototyping", *http://www.mindspring.com/~ggmiller/*~ggmiller/hansen.pdf , 2002.
[IEEE 1998]	IEEE, "IEEE Recommended Practice for Software Requirements Specifications," *IEEE Computer Society, STD-830-1998*, 1998.
[Jacobson 1992]	I. Jacobson, *Object-oriented software engineering : a use case driven approach*. New York; Wokingham, Eng.;Reading, Mass.: ACM Press ;Addison-Wesley Pub., 1992.
[Jacobson 2003]	I. Jacobson, "Use Cases -- Yesterday, Today, and Tomorrow," *The Rational Edge*, 2003.
[Kruchten 2000]	P. Kruchten: *The Rational Unified Process – an Introduction,* Addison-Wesley, 2000.
[Kulak 2000]	D. Kulak and E. Guiney, *Use Cases: Requirements in Context*, ACM Press, 2000.
[Leigh 1996]	A. Leigh, T. Read, N. Tilley, "Problem Oriented Policing: Brit Pop" *Police Research Series, Paper 75,* Home Office, London, 1996.
[Lilly 2000]	S. Lilly, "How to Avoid Use-Case Pitfalls," *Software Development Magazine*, 2000.
[Moisiadis 1998]	F. Moisiadis "A Framework for Prioritizing Use Cases", *http://www.jrcase.mq.edu.au/caise98.html,*JRCASE , 1998.
[O'Connor 2003]	T. O'Connor, "Police Structure and Organization", *http://faculty.ncwc.edu/toconnor/polstruct.htm,* 2003.
[Probasco 2000]	L. Probasco, "The Ten Essentials of RUP," *The Rational Edge*, 2000.
[Rosenberg 2001]	D. Rosenberg and K. Scott, "Driving Design: The Problem Domain," *Software Development Magazine*, 2001.
[Rui 2003]	K. Rui and G. Butler, "Refactoring use case models: the metamodel”," proceeding of the twenty-sixth Australasian computer science conference on Conference in research and practice in information technology - Vol 16., Adelaide, Australia, 2003, pp. 301-308.
[Scott 2000]	M.S. Scott, *Problem-Oriented Policing : Reflections on the first 20 years* , , U.S. Department of Justice, Office of Community Oriented Policing, Washington, 2000
[Sindre 2001]	G. Sindre, A. L. Opdahl, "Template for Misuse Case Description", *Seventh International Workshop on Requirements Engineering: Foundation for Software Quality* , 4-5 June 2001.