in Proceedings of the 16th Intern. Symposium of the International Council on Systems Engineering (INCOSE'06), Orlando, FL, USA, Jul 2006.

The FAR Approach – Functional Analysis/Allocation and Requirements Flowdown Using Use Case Realizations

Magnus Eriksson[1,2], Kjell Borg[1], Jürgen Börstler[2]

[1]BAE Systems Hägglunds AB
SE-891 82 Örnsköldsvik
Sweden
{magnus.eriksson, kjell.borg}@baesystems.se

[2]Umeå University
SE-901 87 Umeå
Sweden
{magnuse, jubo}@cs.umu.se

Abstract. This paper describes a use case driven approach for functional analysis/allocation and requirements flowdown. The approach utilizes use cases and use case realizations for functional architecture modeling, which in turn form the basis for design synthesis and requirements flowdown. We refer to this approach as the FAR (*Functional Architecture by use case Realizations*) approach. The FAR approach is currently applied in several large-scale defense projects within BAE Systems Hägglunds AB and the experience so far is quite positive. The approach is illustrated throughout the paper using the well known Automatic Teller Machine (ATM) example.

INTRODUCTION

Organizations developing software intensive defense systems, for example vehicles, are today faced with a number of challenges. These challenges are related to the characteristics of both the market place and the system domain.

- Systems are growing ever more complex, consisting of tightly integrated mechanical, electrical/electronic and software components.
- Systems have very long life spans, typically 30 years or longer.
- Due to reduced acquisition budgets, these systems are often developed in relatively short series; ranging from only a few to several hundred units. Furthermore, they are often part of a product line of related systems; however, they are always customized for specific customer needs.

For an organization to be competitive in a market like this, it is important to achieve effective development as well as effective maintenance. This makes it necessary to avoid multiple implementations of the same functionality and to achieve high levels of reuse. It is furthermore very important that the different engineering disciplines involved in development can communicate effectively. Development processes should be tightly integrated to ensure that all disciplines are working towards a common goal.

Shortcomings of Traditional Systems Engineering. We believe that systems engineering is a key function in any organization trying to address such complexity. Unfortunately, according to our experience, traditional systems engineering methods and tools suffer from two major shortcomings regarding these challenges:

1. Traditional systems engineering typically utilizes classical structured analysis techniques (Lykins, 2000). In our opinion, these techniques provide little support for achieving high

levels of reuse.

2. Traditional systems engineering utilizes modeling techniques, that are not commonly known to non-technical stakeholders or even other engineering disciplines, like for example IDEFØ diagrams (NIST, 1993). This is a problem, since systems engineering is the means by which stakeholder needs are translated and communicated to other engineering disciplines.

An interesting approach to address the first shortcoming could be to adopt an object oriented approach, like for example the Object Oriented Systems Engineering Method - OOSEM (Lykins, 2000). Although object orientation provides stronger means than structured analysis for achieving high levels of reuse it does not resolve the second shortcoming. Since most approaches are based on the UML (OMG, 2005), they are as unsuitable for our needs as structured analysis. Adopting an UML based approach would certainly ease communications with the software engineering team, but hardly with other stakeholders, like customers, end users, marketing, mechanical and electrical engineering, etc.

Our experience has however shown that one object oriented technique, namely use case modeling (Adolph, 2003), produces outputs that can be easily communicated to both non-technical stakeholders and other engineering disciplines. We therefore propose a use case driven systems engineering method to address the shortcomings of traditional systems engineering mentioned above.

Use Case Modeling. Use case modeling is a functional decomposition technique that provides a semi-formal framework for structuring the system functionality into user goals. These user goals are further specified by a number of scenarios describing the interaction between a system and its actors (users and environment) with the purpose of achieving these goals. The concept of use cases has today become a widely accepted and used requirements modeling technique in the software engineering community. Use case modeling has also started to gain some acceptance within the systems engineering community. At the 2005 INCOSE symposium, for example, Daniels et al. discuss how system level requirements can be derived from use case scenarios (Daniels, 2005). We have however not yet been able to find a systems engineering approach in the literature where use cases are an integral part of the systems engineering process, rather than a simple tool for requirements capture. We have therefore developed a method for functional analysis/allocation and requirements flowdown that utilizes use cases and use case realizations (Kruchten, 2000). We refer to this method as the FAR (*Functional Architecture by use case Realizations*) approach.

Functional Analysis and Allocation. The purpose of the functional analysis and allocation activity is to clearly describe the system functionality, divide functions into sub-functions and to allocate them appropriately to subsystems (INCOSE, 2004). As shown in Figure 1, functional analysis and allocation is based on input provided by requirements analysis, which also receives feedback via the requirements loop from functional analysis and allocation. Furthermore, functional analysis and allocation also provides input to design synthesis and receives feedback through the design loop.

Requirements Flowdown. After specifying a system's functional and physical architecture and allocating system level requirements to subsystems, the next step of the systems engineering process is commonly referred to as Requirements flowdown. The flowdown activity consists of deriving requirements specifications for each element of the systems architecture based on the allocated system requirements (Dorfman, 1997).

Contributions. The main contributions of this paper is a description of how use cases and use case realizations can be utilized for functional architectural modeling and thereby form the basis for system design synthesis and requirements flowdown. The FAR approach is illustrated throughout the paper using the well known Automatic Teller Machine (ATM) example.

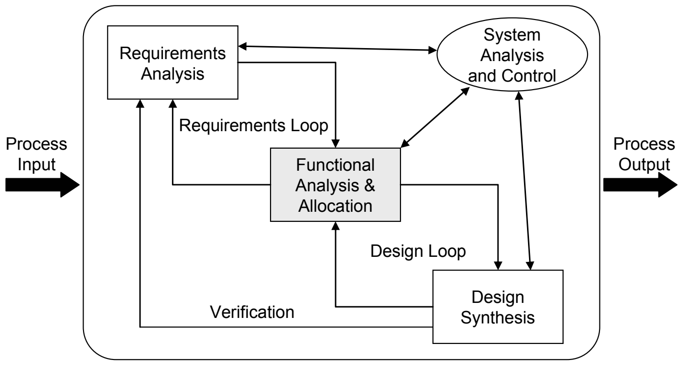
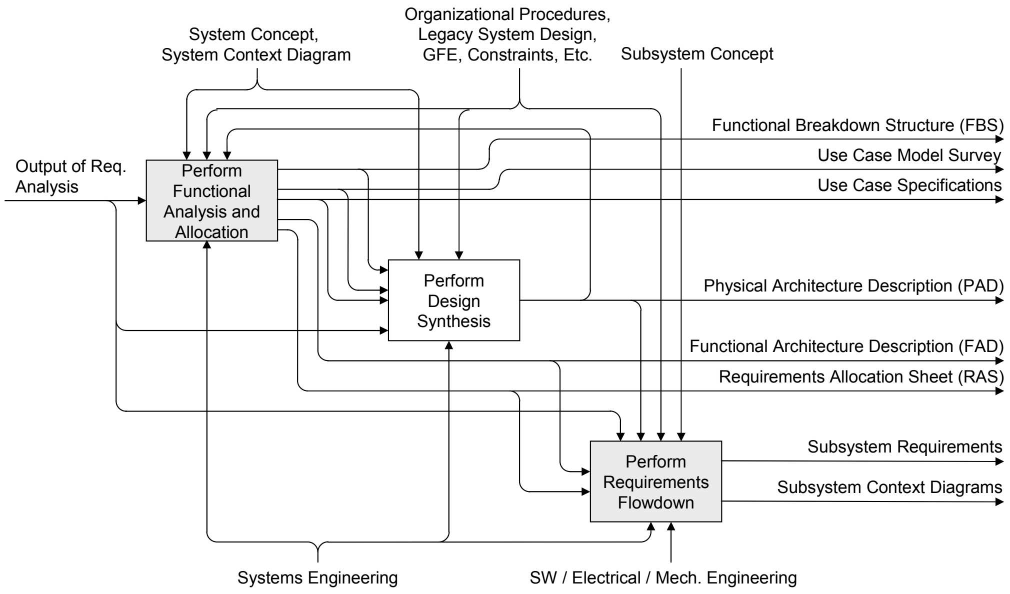


Figure 1. Overview of the systems engineering process (DoD, 2001).

THE FAR APPROACH

Being a use case driven method, FAR closes the requirements loop by producing a use case model as output of the functional analysis and allocation function. As shown in Figure 2, this use case model consists of a survey of all use cases included in the system and detailed specifications of each use case within that survey. Furthermore, in addition to a traditional use case model, the FAR approach also produces a Functional Breakdown Structure (FBS). The purpose of this FBS is to provide a good overview of the system functionality, which we feel is missing in traditional use case models. The produced use case model also serves as input to the Design Synthesis, as shown in Figure 2, and is thereby also part of the design loop shown in Figure 1.

The feedback loop from Design Synthesis to Functional Analysis and Allocation includes a description of the physical architecture of the system. This Physical Architecture Description (PAD) includes, for example, brief descriptions of all defined subsystems. However, details regarding development of this PAD is not within the scope of this paper and will therefore not be further discussed. As shown in Figure 2, the physical architecture enters the functional analysis and allocation function as a control. This control triggers allocations which in turn will result in what we refer to as the Functional Architecture Description (FAD) and a Requirements Allocation Sheet (RAS). A FAD is a collection of all use case realizations for a system, where a use case realization is a description of how different subsystems collaborate to solve a specific use case (Kruchten, 2000). As shown in Figure 2, the FAD and the RAS are input to the requirements flowdown activity, which produce requirements specifications for different subsystems as output.



System Concept, System Context Diagram
Organizational Procedures, Legacy System Design, GFE, Constraints, Etc.
Subsystem Concept
Output of Req. Analysis
Perform Functional Analysis and Allocation
Functional Breakdown Structure (FBS)
Use Case Model Survey
Use Case Specifications
Perform Design Synthesis
Physical Architecture Description (PAD)
Functional Architecture Description (FAD)
Requirements Allocation Sheet (RAS)
Perform Requirements Flowdown
Subsystem Requirements
Subsystem Context Diagrams
Systems Engineering
SW / Electrical / Mech. Engineering

Figure 2. Context diagram for the FAR approach in IDEFØ.

System Context Diagram. One, according to our experience, important artifact to develop and agree upon before starting functional analysis and allocation is a system context diagram, see Figure 3. The system context diagram clearly visualizes the system-of-interest (ISO/IES, 2002), its external interfaces and its users. It defines the set of allowed entities for describing the black-box functional view of the system. This helps analysts maintaining focus on functions within the scope of the system. As show in Figure 3, our system context diagrams contain less information than for example the "elaborated system context diagram" used in OOSEM (Lykins, 2000) or an IDEFØ context diagram (NIST, 1993). By omitting details regarding provided services and input/output data, we improve the readability of the diagram.

A use case actor is some type of entity, external to the system-of-interest, which can interact with the system by exchanging information (OMG, 2005). A use case actor can either be a human user or an external system. A use case actor can have association relationships to use cases, and generalization relationships to other actors. An association relationship to a use case means that the actor either initiates the use case or takes part in performing the behavior defined by the use case. If an actor can initiate a specific use case, it is a primary actor of that use case, else a secondary actor. In FAR, generalization relationships between actors are shown as closed head arrows between actors in the system context diagram, as illustrated in Figure 3. A generalization means that the more specialized actor inherits all association relationships from the more general actor. It can communicate with all use cases as the more general actor, but it can also have association relationships to further use cases. In the FAR approach, this relation is utilized for creating groups of users that can initiate a common set of use cases. We have chosen to include these relationships between actors in our system context diagram so that it may provide a full overview of all entities that can interact with the system-of-interest.

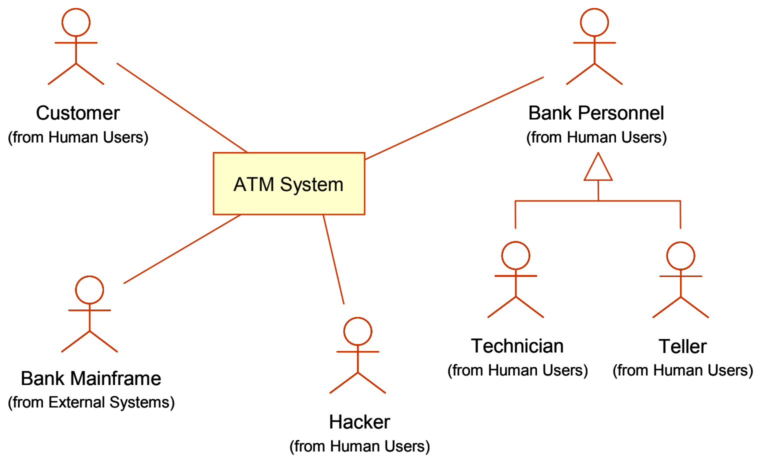
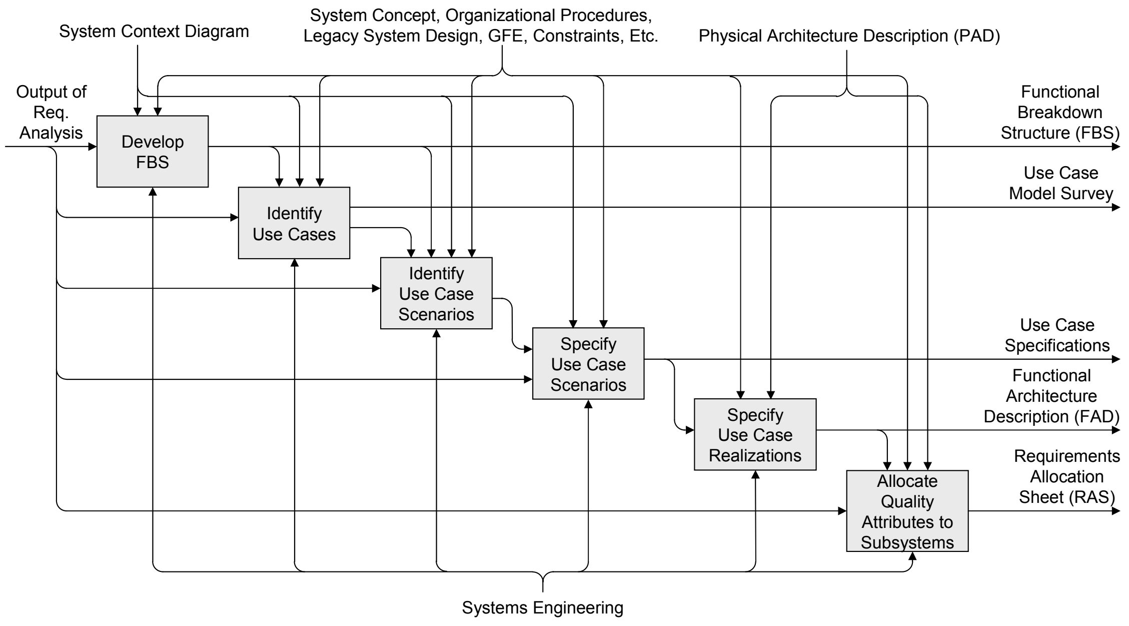


Figure 3. System context diagram example for an ATM system.

The "*Perform Functional Analysis and Allocation*" Function

As shown in Figure 4, the Perform functional analysis and allocation function is decomposed into six sub-functions. The first four of these sub-functions regard functional analysis by development of a use case model. The last two sub-functions regard allocation of functions and quality attributes to subsystems. In the following subsections, we will discuss these different sub-functions and the different modeling artifacts they result in.



System Context Diagram
System Concept, Organizational Procedures, Legacy System Design, GFE, Constraints, Etc.
Physical Architecture Description (PAD)
Output of Req. Analysis
Develop FBS
Identify Use Cases
Identify Use Case Scenarios
Specify Use Case Scenarios
Specify Use Case Realizations
Allocate Quality Attributes to Subsystems
Functional Breakdown Structure (FBS)
Use Case Model Survey
Use Case Specifications
Functional Architecture Description (FAD)
Requirements Allocation Sheet (RAS)
Systems Engineering

Figure 4. The "*Perform Functional Analysis and Allocation*" function in IDEFØ.

The "*Develop FBS*" Function. The first step of the functional analysis is to develop a preliminary FBS based on the output from the requirements analysis activity. As shown in Figure 5, this FBS should include all major function groups and services provided by the system-of-interest.

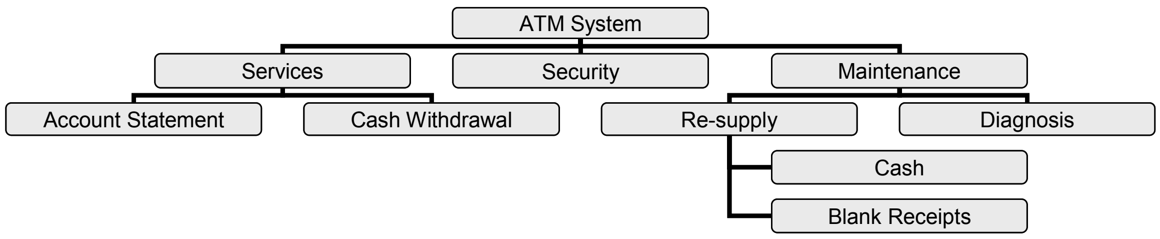


Figure 5. Example FBS for an ATM system.

The "*Identify Use Cases*" Function. Our way of working with use cases has to a large extent been inspired by the work on *Patterns for Effective Use Cases* by Adolph et al. (Adolph, 2003), in particular their goal oriented approach to use case modeling. A use case must either be a complete goal of the primary actors of that use case or a sub-goal derived from another use case (see Figure 6). Complete-goal use cases shall be derived from and be traceable to system requirements. Included sub-goal use cases are typically identified by finding common behavior in several complete-goal use cases that can be extracted and referenced, and thereby only be specified once in the model. Use cases are named as verb-phrase-goals and depicted in UML use case diagrams (OMG, 2005) as shown in Figure 6.

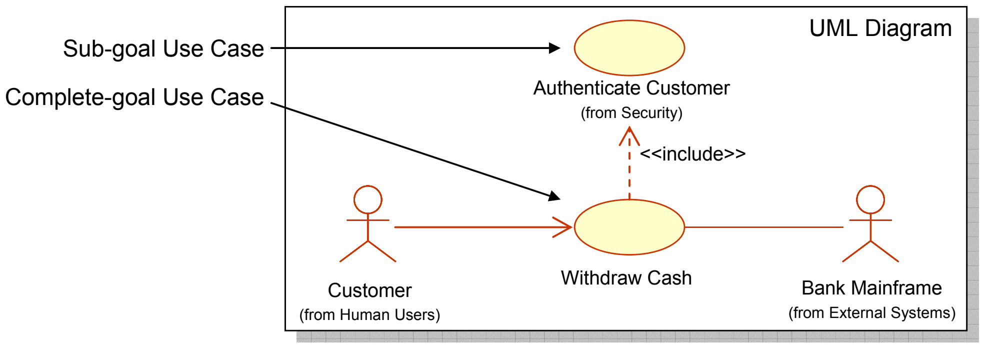


Figure 6. An example use case decomposition.

A difference between the FAR approach and traditional use case modeling is that we do not use UML use case diagrams to provide an overview over system functionality and system scope; we use a FBS for this purpose. We utilize use case diagrams only to visualize relations between individual use cases and to provide an overview of the primary and secondary actors of each use case. We provide this actor overview by using directed associations for primary actors and undirected associations for secondary actors of each use case, as shown in Figure 6. These individual use case diagrams are then linked to their corresponding elements in the FBS. For example, the use case diagram in Figure 6 would be linked to the "Cash Withdrawal" element in the FBS in Figure 5. Furthermore, there would also be another use case diagram describing the details regarding the sub-goal use case "Authenticate Customer" which would be linked to the "Security" element of the FBS, as indicated by the parenthesis under the use case name in Figure

6.

By tagging suitable sub-trees of the FBS as use case packages (Kruchten, 2000), and utilizing links between the FBS and use case diagrams, a “Use Case Model Survey” for the system can be created. This survey contains the following elements:

- A system context diagram.
- Descriptions of all use case actors.
- An overview picture of the elements of the FBS which are tagged as use case packages. These elements form the use case model hierarchy and will therefore also have a corresponding heading in the survey outline.
- UML use case diagrams for all use case packages.
- Brief descriptions of all use cases.

The “*Identify Use Case Scenarios*” Function. Each use case is described by one or more scenarios. Like Adolph et al. (Adolph, 2003), we distinguish between the following three types of scenarios:

- A “*Main Success Scenario”*, which describes the “normal” way of achieving the goal stated in the use case name.
- “*Alternative Scenarios*”, which describe alternative ways of achieving the goal stated in the use case name.
- “*Exceptional Scenarios*”, which describe how different failures are detected and handled by the system.

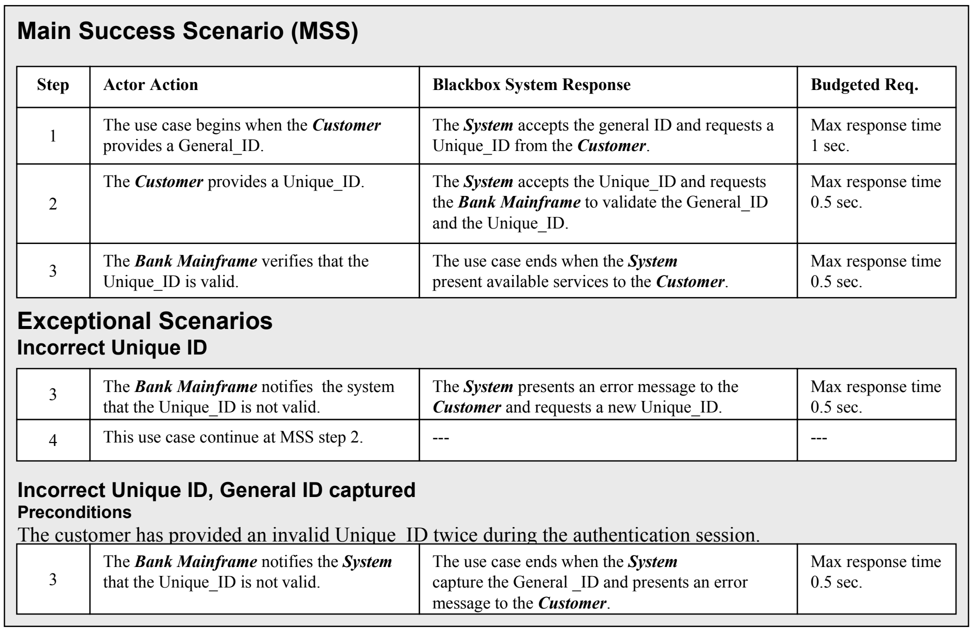
Each of these scenarios should be named in a way that clearly describes what is unique about it. The only exception from this naming rule is the main success scenario, which is not named if the normal procedure is obvious from the scenario context or use case name.

The “*Specify Use Case Scenarios*” Function. We describe use case scenarios in a tabular natural language notation. This notation is based on the RUP-SE black-box flow of events (Rational, 2003), which we consider to have three major advantages compared to traditional textual descriptions:

1. The notation is slightly more formalized and thereby easier to manage using tools.
2. The notation has separate fields for actor actions and system responses to those actions. This simple fact forces analysts to always think about interfaces, which is a key success factor for maintaining focus in the modeling of complex systems.
3. The “Blackbox Budgeted Requirements” field (rightmost column in Figure 7) of the notation provides an intuitive way to allocate quality attributes to a use case scenario, which is not present in traditional notations.

As illustrated in Figure 7, we describe alternative and exceptional scenarios as so called scenario fragments (Adolph, 2003). These scenario fragments are specified as deltas to the main success scenario. The first step identifier of the fragment determines the position where the fragment should be inserted into the main success scenario. A fragment can either return to the main success scenario, as “*Incorrect Unique_ID*” in Figure 7, or terminate as “*Incorrect Unique_ID, General_ID captured*” in Figure 7.

Besides the actual tabular scenarios, our use case specifications also include introductory information such as context information regarding the use case goal and possible pre-conditions and post-conditions. It is also possible to add this type of information to individual scenarios, if needed, as shown in scenario “*Incorrect Unique_ID, General_ID captured*” in Figure 7.



Main Success Scenario (MSS)

Step	Actor Action	Blackbox System Response	Budgeted Req.
1	The use case begins when the ***Customer*** provides a General_ID.	The ***System*** accepts the general ID and requests a Unique_ID from the ***Customer***.	Max response time 1 sec.
2	The ***Customer*** provides a Unique_ID.	The ***System*** accepts the Unique_ID and requests the ***Bank Mainframe*** to validate the General_ID and the Unique_ID.	Max response time 0.5 sec.
3	The ***Bank Mainframe*** verifies that the Unique_ID is valid.	The use case ends when the ***System*** present available services to the ***Customer***.	Max response time 0.5 sec.

Exceptional Scenarios

Incorrect Unique ID

3	The ***Bank Mainframe*** notifies the system that the Unique_ID is not valid.	The ***System*** presents an error message to the ***Customer*** and requests a new Unique_ID.	Max response time 0.5 sec.
4	This use case continue at MSS step 2.	---	---

Incorrect Unique ID, General ID captured

Preconditions

The customer has provided an invalid Unique_ID twice during the authentication session.

3	The ***Bank Mainframe*** notifies the ***System*** that the Unique_ID is not valid.	The use case ends when the ***System*** capture the General _ID and presents an error message to the ***Customer***.	Max response time 0.5 sec.

Figure 7. Example black-box scenario descriptions.

Even though we have chosen to exclude feedback loops to improve the readability of Figure 4, development of these types of use case models is highly iterative. Identifying use cases and linking these to the FBS, as well as identification of alternative and exceptional use case scenarios, will typically result in the identification of new or modified elements in the FBS. Furthermore, describing use case scenarios, often results in new use cases being discovered and other use cases are being merged. The reasons for this type of restructuring of the use case model are usually to ensure that certain rules of thumb regarding use case quality are fulfilled. Examples of such rules are:

- A main success scenario should have between three and ten steps.
- There should not be common behavior described in several use cases. Common behavior should be broken out as a sub-goal use case and referenced by the source use cases.
- Scenarios should not have complex control structures such as nested loops or if-statements. If such are needed, a use case should rather be divided into sub-goal use cases to maintain high readability.

The "*Specify Use Case Realizations*" Function. Development of FAR use case realizations is in essence the processes of allocating functions, modeled as use cases, to subsystems defined by the system's physical architecture. As shown in Figure 8, our notation for use case realizations is based on the RUP-SE white-box flow-of-events (Rational, 2003). Each black-box step of a use case scenario is decomposed into a number of so called white-box steps. These white-box steps describe how different subsystems collaborate to solve each use case scenario black-box step. During this process, each "Blackbox Budgeted Requirement" must be decomposed into a number of "Whitebox Budgeted Requirements". Development of these use case realizations usually involves trade-off analyses to evaluate different allocations, considering for example quality attributes such as availability, reliability, cost, etc.

Incorrect Unique ID, General ID captured
Preconditions
The customer has provided an invalid Unique_ID twice during the authentication session

Step	Actor Action	Blackbox System Response	Budgeted Req.	Whitebox Action	Whitebox Budgeted Req.
3	The ***Bank Mainframe*** notifies the ***System*** that the Unique_ID is not valid.	The use case ends when the ***System*** capture the General_ID and presents an error message to the ***Customer***.	Max response time 0.5 sec.	The ***MainframeInterface*** accepts the result of the authentication attempt from the ***BankMainframe*** and notifies the ***TransactionControler*** that the Unique_ID is not valid	Max response time 0.1 sec.
				The ***TransactionControler*** requests the General_ID to be captuered by the ***CardReader***.	Max response time 0.2 sec.
				The ***TransactionControler*** requests an error message to be presented by the ***OutputDevice***.	Max response time 0.1 sec.
				The ***OutputDevice*** presents the error message to the ***Customer***.	Max response time 0.1 sec.

Figure 8. An example use case realization.

The "*Allocate Quality Attributes to Subsystems*" Function. Not all system requirements are suitable for use case modeling. System-wide quality attributes, like environmental constraints, legislations, standards, etc. are such examples. These requirements must be handled in parallel to requirements modeled as use cases. We manage these requirements in a Requirements Allocation Sheet (RAS). A RAS documents which subsystems are involved in fulfilling each requirement. When performing this activity it is important to capture background information and rationales for allocations. As shown in Figure 9, we also utilize the RAS to maintain traceability between system requirements and use cases.

○ Use cases
○ Allocation
● UC &Allocation

		Use Cases					Subsystems						Allocation Rationale
Req. ID	**Req. Text**	Withdraw Cash	Get Account Statement	Authenticate Customer	…	…	Mainframe Interface	Transaction Controler	Card Reader	Output Device	…	…	
Services													
SYSR867	The system shall provide a cash withdrawal service.	X											
SYSR868	The system shall provide a account statement service.		X										
…	…												
Security													
SYSR870	The system shall capture the General_ID after three consecutive failed authentication attempts.			X									
…	…												
Maintenance													
SYSR900	It shall be possible to replace any malfunctioning subsystem within 15 minutes of…						X	X	X	X	X	X	Some rationale for the allocation…
…	…												

Figure 9. An example requirements allocation sheet.

The "*Perform Requirements Flowdown*" Function

In the FAR approach, specifying subsystem requirements is the responsibility of domain engineers, see Figure 10. This is in contrast to the traditional view, where systems engineering is responsible for this activity. Experience has shown that passing specifications between for example systems- and software engineering does not yield satisfactory results (INCOSE, 2004). By allocating this responsibility to domain engineers, we force them to analyze the origin of subsystem requirements, which will give them a better system understanding. It is however critical that systems engineering is part of this process to ensure that the resulting subsystem requirements still represent the intent of the original system level requirements and their corresponding allocations.

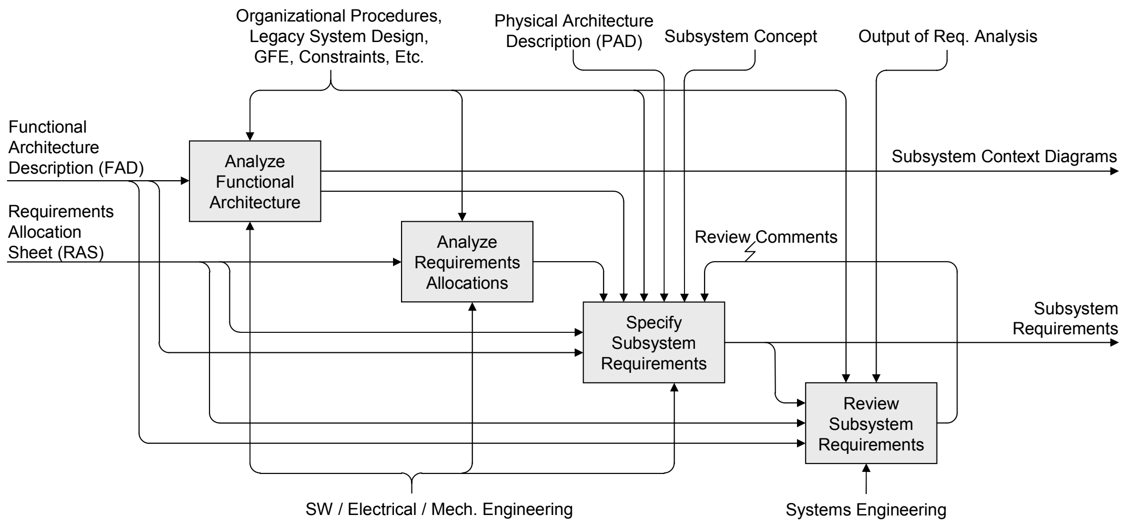

Organizational Procedures, Legacy System Design, GFE, Constraints, Etc.
Physical Architecture Description (PAD)
Subsystem Concept
Output of Req. Analysis
Functional Architecture Description (FAD)
Requirements Allocation Sheet (RAS)
Analyze Functional Architecture
Analyze Requirements Allocations
Subsystem Context Diagrams
Review Comments
Specify Subsystem Requirements
Subsystem Requirements
Review Subsystem Requirements
SW / Electrical / Mech. Engineering
Systems Engineering

Figure 10. The "*Perform Requirement Flowdown*" function in IDEFØ.

The "*Analyze Functional Architecture*" and the "*Analyze Requirements Allocations*" Functions. These activities serve as controls to enable specification of subsystem requirements. During these activities also subsystem context diagram for each of the defined subsystems are developed. However, in contrast to the previously mentioned system context diagram, these diagrams visualize the internal, rather than the external, interfaces of a system. This means that the use case actors of a subsystem context diagram will typically not be system users, but rather other subsystems.

Analysis of the FAD, focus on identifying logical interfaces between subsystems, and allocation of responsibilities for sub-functions. It is important that FAD white-box steps are analyzed in the correct context. Without correct sequence-, and other background information such as possible pre-conditions, analysis may yield invalid results. Analysis of a RAS consists to a large extent of understanding the rationale(s) behind each allocation and the effect of every allocation for a particular subsystem.

The "*Specify Subsystem Requirements*" Function. The purpose of this activity is to produce subsystem requirements that later will form the basis for detailed design. The analysis of the FAD should result in one or more subsystem requirements being specified for each white-box step. As shown in Figure 11, these subsystem requirements are typically of input, output or functional character Analysis of the RAS will result in one or more subsystem requirements being specified for each ‘X’ in the “Subsystems” part of the matrix. These resulting subsystem requirements are usually quality attributes constraining the development of the subsystems.

Incorrect Unique ID, General ID captured
Preconditions
The customer has provided an invalid Unique_ID twice during the authentication session

Whitebox Action	Whitebox Budgeted Req.	Subsystem Requirement
The ***TransactionControler*** requests the General_ID to be captuered by the ***CardReader***.	Max response time 0.2 sec.	The ***TransactionControler*** shall produce a request to capture a General_ID within 0.15 sec, if an invalid Unique_ID have been provided three times.
		The ***CardReader*** shall accept requests to capture a General_ID within 0.05 sec.
		The ***CardReader*** shall capture a General_ID on request.

Figure 11. Example of subsystem requirements derived from use case realizations.

The “*Review Subsystem Requirements*” Function. Although the responsibility of specifying subsystem requirements is allocated to domain engineers, systems engineering must still verify that the derived subsystem requirements represent the original intent of the system level requirements. The resulting subsystem requirements must therefore be reviewed by systems engineering, before they can be baselined.

Managing Scope Creep

It is often hard to draw a sharp line between fulfilling customer expectations and “gold plating” a system. Also preparing a system architecture for anticipated changes, may result in many additional internal requirements. These “prepared for” requirements are often unclear and can have a significant impact on the system architecture. It is therefore very important to keep track of them.

In the FAR approach, “gold plating” and “prepared for” requirements are modeled by means of so-called change cases. Change cases, first proposed by Ecklund et al. (1996), are basically use cases that specify anticipated changes to a system. Each change case has “impact links” to all use cases that are affected, when the change case is realized (see Figure 12). These impact links provide a good basis for change impact analysis. In FAR, change cases are used to mark proposed, but not yet accepted functionality. Whenever there is a discussion if a function is gold plating or not, the function should be marked as a change case. However, once accepted for implementation within the system, these change cases are transformed to use cases.

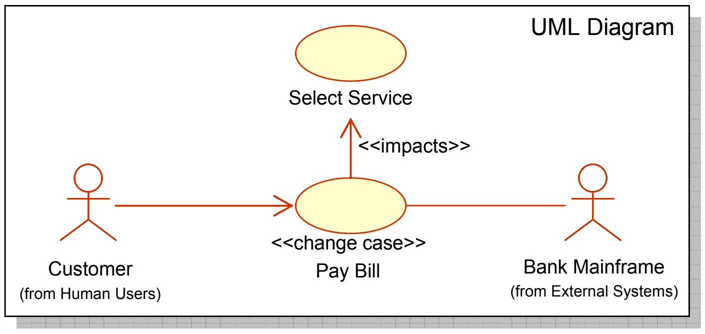


Figure 12. An example change case.

Managing the Level of Detail

There is no straight answer to the question "How much systems engineering is enough?" A decision to baseline the systems engineering output and move on to detailed design is a matter of risk management. A better question would be "On what level of specification detail can an organization move on from systems engineering to detailed design with acceptable risks?" In FAR, there are basically two ways to manage this level of specification detail:

1. Reduce or increase the number of subsystems in the design. More subsystems mean more white-box steps and more detail in the FAD and vice versa. The two extremes on this scale would be an architecture comprising a single subsystem (the system itself), or an architecture where each configuration unit is a separate subsystem. In the first case, FAD white-box steps would actually be equal to their respective black-box steps. Basically no design decisions are made. The second case would lead to a totally specified system, where there is no room for consecutive detailed design. Our experience has shown that a number of subsystems leading to typically two to four white-box steps per black-box step result in specifications with high readability.
2. Adjust the level of detail by recursion. Complex/high risk subsystems may be further analyzed and described by moving to the next lower level of abstraction and treating them as the system-of-interest. This enables us to apply the FAR methodology recursively, since the inputs to and outputs of the FAR process are basically the same.

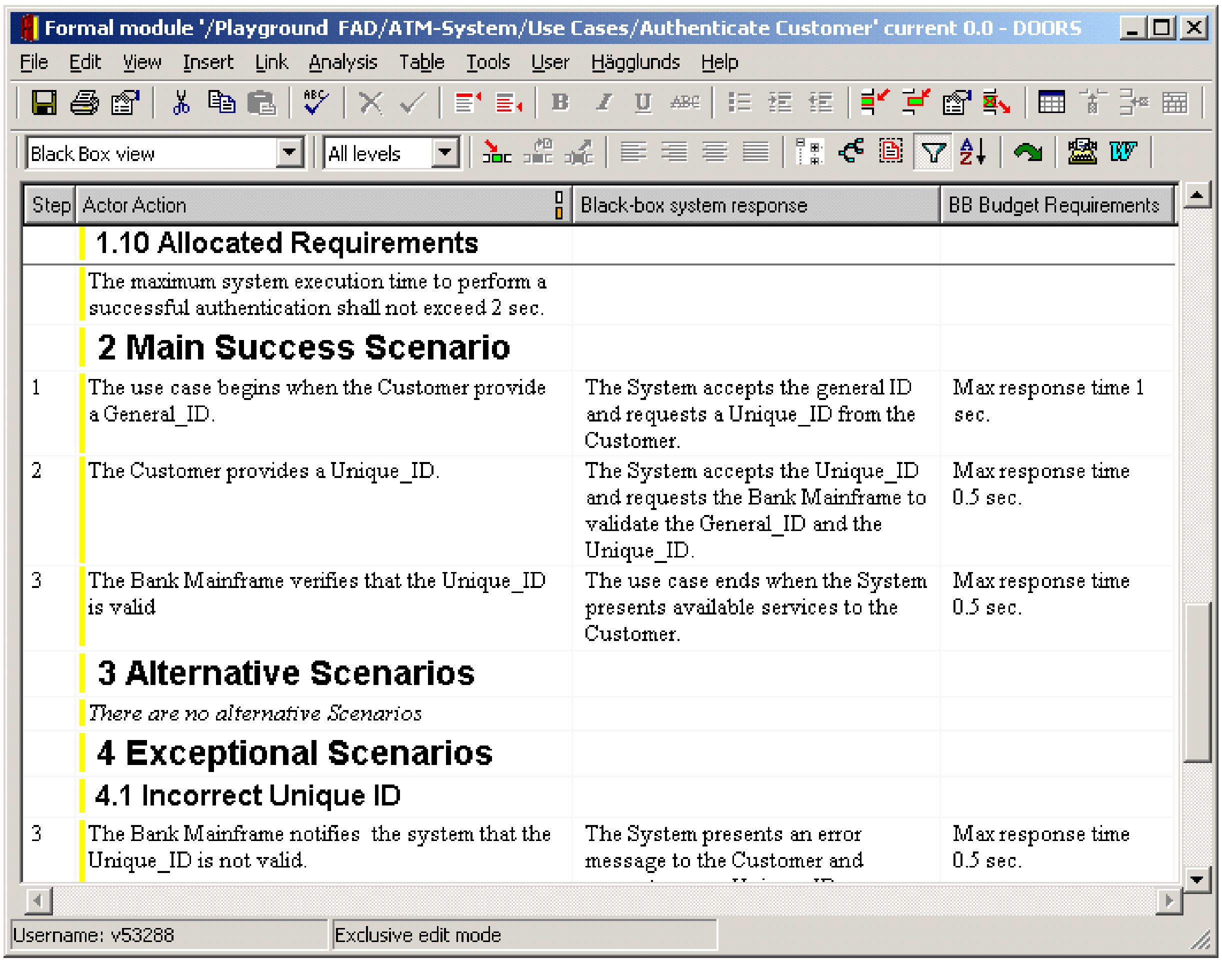
Tool Support for the FAR Approach

Adequate tool support is an important factor when managing models of complex systems. We have chosen to utilize the commercial requirements management tool Telelogic DOORS to manage our FAR models. DOORS maintains a database, where "objects" may be arranged in a hierarchical structure in modules. Typically, node objects are used as headings and leaf objects are used for data items, such as requirements in these modules. DOORS supports a link concept that can be used to define relationships between objects. These links form the basis for traceability in DOORS. It is possible to define attributes on both module level and object level in DOORS. Object attributes can be displayed in columns within a module in the DOORS graphical user interface. Combining this possibility with DOORS' support for filtering on properties of objects, a user can define views suiting different needs. These views are used when working with data and also for reporting and exporting information to other tools.

FBSs are managed in specific DOORS modules. The basic idea is to use the standard

DOORS object hierarchy for building FBS trees. Each FBS element becomes a heading; leaf objects are used for capturing different kinds of information regarding each FBS element. Examples of such information are short descriptions of each element and use case diagrams showing possible use cases linked to each element.

Each use case is managed as a separate module in the DOORS database. Introductory information is captured in a traditional document structure with headings and text. To capture use case scenarios and use case realizations in DOORS, we use one DOORS object to describe each scenario step, as shown in Figure 13. The relationship between black-box and white-box scenario steps (realizations) is managed using the standard DOORS object hierarchy. All white-box step objects are children of their corresponding black-box step object. This means that use case specifications and their corresponding realizations are different views of the same DOORS module.


Formal module '/Playground FAD/ATM-System/Use Cases/Authenticate Customer' current 0.0 - DOORS
File Edit View Insert Link Analysis Table Tools User Hägglunds Help

Black Box view | All levels

Step	Actor Action	Black-box system response	BB Budget Requirements
	1.10 Allocated Requirements		
	The maximum system execution time to perform a successful authentication shall not exceed 2 sec.		
	2 Main Success Scenario		
1	The use case begins when the Customer provide a General_ID.	The System accepts the general ID and requests a Unique_ID from the Customer.	Max response time 1 sec.
2	The Customer provides a Unique_ID.	The System accepts the Unique_ID and requests the Bank Mainframe to validate the General_ID and the Unique_ID.	Max response time 0.5 sec.
3	The Bank Mainframe verifies that the Unique_ID is valid	The use case ends when the System presents available services to the Customer.	Max response time 0.5 sec.
	3 Alternative Scenarios		
	There are no alternative Scenarios		
	4 Exceptional Scenarios		
	4.1 Incorrect Unique ID		
3	The Bank Mainframe notifies the system that the Unique_ID is not valid.	The System presents an error message to the Customer and	Max response time 0.5 sec.

Username: v53288 | Exclusive edit mode

Figure 13. The DOORS use case specification view.

To make work more efficient, we have developed a small number of tool extensions using DOORS' integrated scripting language DXL (Telelogic, 2004).

- A "*Create Use Case Module*" tool, which automatically creates use case modules with certain properties. This tool has been realized by pre-creating a use case template module with predefined structure, attributes and views.
- A "*Create Use Case Model Survey*" tool, which filters out a use case model survey from

a FBS module. The resulting view can then be exported into a Use Case Model Survey Report using the standard DOORS MS Word export.
- An "*Export Use Case*" tool, which exports use case modules from DOORS to MS Word as use case specifications and use case realizations using the FAR notation. The tool is based on the standard DOORS MS Word export. The basic idea of the tool is to export black-box and white-box scenarios as specially formatted tables and all other information as ordinary headings and text. The tool distinguishes between scenarios and other information using the "Step" attribute (leftmost column in Figure 13), which only has a defined value if an object is part of a scenario.

These extensions are a subset of the "PLUSS Toolkit", which is described in detail in (Eriksson, 2005).

SUMMARY AND CONCLUSIONS

We have described how use case modeling can become an integral part of the systems engineering process instead of simply being a tool for requirements analysis. By developing system level use case realizations, the FAR approach extends use case modeling to form the basis for functional allocation as well as requirements flowdown.

The FAR approach is currently applied in several large scale defense projects within BAE Systems Hägglunds AB. Preliminary results indicate that the FAR approach performs well within this organization (Eriksson, 2006).

FUTURE WORK

One important area of future work regarding the FAR approach is improved support for modeling quality attributes, such as for example availability and system safety requirements. One interesting approach for this, which is likely to fit very well in the FAR framework, is referred to as "Misuse Case" modeling (Sindre, 2001). Sindre and Opdhal (2001) define a misuse case as "… the inverse of a use case, a function that the system should not allow." This technique has been applied on for example security requirements (Sindre, 2005), and is likely to also be applicable other types of quality attributes.

ACKNOWLEDGEMENTS

The authors would like to thank Henrik Morast at Syntell AB and all the people at BAE Systems Hägglunds AB that contributed to this work.

REFERENCES

Adolph S., Bramble P., Cockburn A., Pols A., *Patterns for Effective Use Cases*, Addison-Wesley, 2003

Daniels J., Botta R., Bahill T., *A Hybrid Requirements Capture Process*, Proceedings of the 15'th International INCOSE Symposium, 2005

Department of Defense (DoD), *Systems Engineering Fundamentals*, Defense Acquisition University Press, Fort Belvoir, Virginia, 22060-5565, January 2001

Dorfman M., Thayer R. H., *Software Requirements Engineering*, Los Alamitos, CA, IEEE Computer Society Press, 1997

Ecklund E., Delcambre L., Freiling M., *Change Cases - Use Cases that Identify Future Requirements*, Proceedings of OOPSLA'96, San Jose, Ca, October 6-10, 1996, pp. 342-358.

Eriksson M., Morast H., Börstler J., Borg K., *The PLUSS Toolkit – Extending Telelogic DOORS and IBM-Rational Rose to Support Product Line Use Case Modeling*, Proceedings of 20'th IEEE/ACM International Conference on Automated Software Engineering, 2005

Eriksson M., Börstler J., Borg K., *Performing Functional Analysis/Allocation and Requirements Flowdown Using Use Case Realizations – An Empirical Evaluation*, Proceedings of the 16'th International INCOSE Symposium, 2006

International Council on Systems Engineering (INCOSE), *Systems Engineering Handbook Version 2a*, INCOSE-TP-2003-016-02, June 2004

ISO/IEC, *Systems Engineering – System life cycle processes*, International Standard: ISO/IEC FDIS 15288:2002(E), 2002

Kruchten P., *The Rational Unified Process - An Introduction*, Second Edition, Addison-Wesley, 2000

Lykins H., Friedenthal S, Meilich A., *Adapting UML for an Object Oriented Sysyems Engineering Method (OOSEM)*, Proceedings of the 10'th International INCOSE Symposium, 2000

National Institute of Standards and Technology (NIST), *Integration Definition for Function Modeling (IDEF0)*, Federal Information Processing Standards 183, 1993

Object Management Group: Unified Modeling Language Version 2.0, Available at: http://www.uml.org, 2005

Rational Software, *The Rational Unified Process for Systems Engineering - Whitepaper Ver. 1.1*, Available at: http://www.rational.com/media/whitepapers/TP165.pdf, 2003

Sindre G., Opdahl A., *Templates for Misuse Case Description*, Proc. 7th Intl Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001), Interlaken, Switzerland, June 2001

Sindre G., Opdahl A., *Eliciting Security Requirements with Misuse Cases*, *Requirements Engineering*, vol. 10, January 2005

Telelogic AB, *DXL Reference Manual,* DOORS 7.1, Manual creation date: (3 May 2004)

BIOGRAPHY

Magnus Eriksson is a systems engineer with BAE Systems Hägglunds AB in Örnsköldsvik, Sweden. Magnus has, since he joined Hägglunds, been a key contributor in areas such as requirements engineering, system architecture and process improvement. He is currently also active as PhD student at Umeå University in Sweden, from where he also earned his Master of Science degree in engineering. His main research interests are in the areas software product line development and systems engineering.

Kjell Borg is currently software quality manager at BAE Systems Hägglunds in Örnsköldsvik, Sweden, and has over 20 years of experience in software engineering and research. He holds a BSc in Mathematics and a PhLic in Computer Science from Umeå University in Sweden. Mr Borg has experience in the fields of Usability Design, Requirements Engineering, Software Engineering, Quality Management, Embedded Systems, Project Management and Systems Engineering. He has business experience, both as an employee and consultant, in areas of Telecom, Defence, Transport, Process Industry, etc.

Jürgen Börstler is an associate professor and director of studies in computing science at Umeå University, Umeå, Sweden. He received a Masters degree in Computer Science, with Economics as a subsidiary subject, from Saarland University, Saarbrücken, Germany and a PhD in

Computer Science from Aachen University of Technology, Aachen, Germany. His main research interests are in the areas object-oriented software development, requirements management, process improvement, and educational issues in computer science.