# Case study of model-based systems engineering (MBSE): Part 1. The centralized systems model of IBM Rational Harmony

Mohit Choudhary November 15, 2011

Modeling has been an important part of systems engineering since its inception. During the last decade, engineers have significantly increased their use of model-based technologies to evolve a new discipline of Model-Based Systems Engineering (MBSE). This discipline differs from traditional systems engineering in that it emphasizes a central system model that captures both system requirements as well as the design decisions that fulfill them. In addition to serving as a knowledge repository for systems engineering work artifacts, the system model can also be simulated to validate cost or performance studies and design choices. Highly practiced MBSE processes like IBM Rational Harmony for Systems Engineers focus on system functional analysis, which is the translation of functional requirements into a coherent description of system operations. The system operations are then used to derive ports and interfaces among the allocated system architecture blocks. These interfaces form the basis of the formal hand-off among various sub-systems.

View more content in this series

This part of our series aims to study the standard MBSE process through the use of a case study. First, we formulate the scope of the case study based on the design of a UAV ground station controller. Then we introduce the basic concepts, task flow and work products of the Rational Harmony systems engineering process. Finally, we go through the defined task flow to arrive at the design of the UAV ground station controller while constructing the required artifacts in each phase.

## The case study

This case study is based on the design analysis of a small part of a UAV ground station controller that must meet the requirements in Table 1.

### Table 1. UAV ground station requirements

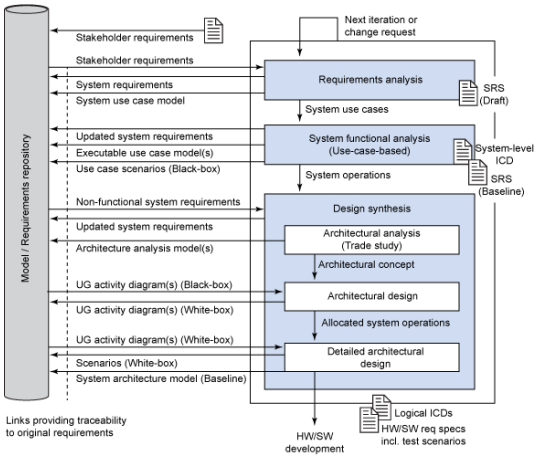
| Requirement Reference | Requirement |
|---|---|
| 01 | Access information of UAVs in flight. (Identity and sensor payload) |
| 02 | Allow operator to assign search areas to a selected UAV in flight. |

| | |
|---|---|
| 03 | Receive sensor track information from UAV at a frequency of 1 update /sec. |
| 04 | Maintain a 30min history of tracks in the system. |
| 05 | Allow the operator to maintain a repository of adopted System tracks. |
| 06 | Maintain a maximum of 100 System Tracks. |
| 07 | Allow the operator to perform lifecycle operations on system track (create/ drop). |
| 08 | Update the System track per second with the primary sensor track update if available else with DR'ed value. |
| 09 | Make the system tracks available on the display and plot their updates. |
| 10 | Allow the operator to perform operator assisted system track association with another UAV's sensor track. |
| 11 | Allow the operator to merge two separate system tracks into one. |
| 12 | Alert the operator of important events in the system like creation and dropping of System track. |
| 13 | Allow the operator to abort the search of a UAV at any time. |

# Model-based system engineering in Rational Harmony systems engineering

Rational Harmony for Systems Engineering enables you to identify and derive the required system functions, and also identify associated system modes and states. In addition, you can allocate the identified system functions and states to a sub-system structure and identify the ports and interfaces across sub-systems. Figure 1 shows essential inputs and outputs of each engineering phase that you must perform in order to arrive at a system design.

## Figure 1. Life cycle of the engineering phases




During the functional analysis phase, the functional flow of a use case is defined through an activity diagram. Then, use case scenarios are derived from the activity diagram. The scenarios are represented by a set of sequence diagrams, which are required to create ports and interfaces of the use case block. Lastly, state-based behavior of the use case is captured in a state chart diagram.

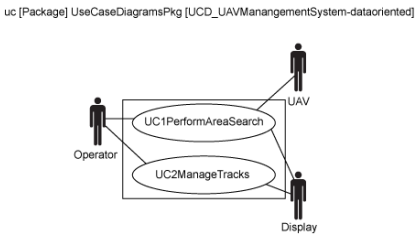
During the architectural design phase, the chosen system block is decomposed into parts. The resulting system structure is captured in a SysML block definition diagram (BDD) and in a SysML internal block

Diagram (IBD). The allocation can be graphically represented for each use case through an associated white box activity diagram. This diagram is a copy of the use case black box activity diagram, but it is partitioned into swim lanes. Each lane represents a block of the decomposition hierarchy. Based on the chosen design concept, the operations are then "moved" into respective block swim lanes. An essential requirement for this allocation is that the initial links (functional flow) between the actions are maintained. Finally, the focus of the detailed architectural design phase is on the definition of ports and interfaces, as well as on the definition of the state-based behavior of the system blocks at the lowest level of the architectural decomposition.

## The design process

The system requirements of the UAV ground station are grouped into two use cases as shown in Figure 2. For traceability, the identified system requirements are linked to the associated use cases. For this article, assume that the requirement analysis is complete. For this case study, we will focus on the Uc1PerformAreaSearch use case.

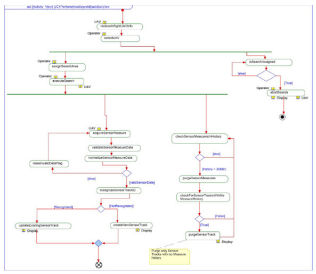
### Figure 2. UAV management system use case diagram




## Functional analysis

The functional flows of the use case cover the aspects of assigning a search to a selected UAV, receiving the track information from the UAV sensor, maintaining the same in the system as sensor track, maintaining the required history of sensor track updates and finally allowing the operator to abort the search. You can use the tool to elaborate each of the functional flows in a black box activity diagram as shown in figure 3.

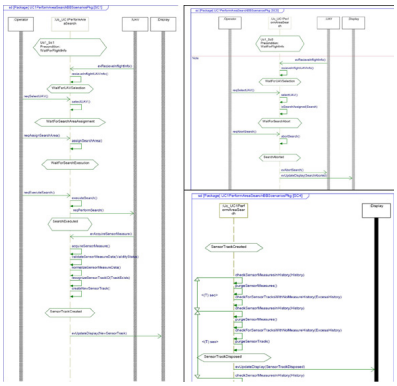
### Figure 3. Black box activity diagram



### Use case scenarios

You can see that each flow in the activity diagram represents a different use case scenario. These flows not only help us in detailing the operations in a functional flow, but also form the basis of validating use case behavior through the various stages of development. Three of the five scenarios that you can derive for our use case are shown in Figure 4.

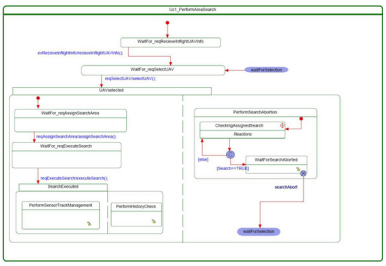
### Figure 4. Black box use case scenarios



### Use case state diagram

In the next step you can derive the ports and interfaces by using the sequence diagrams. After getting the ports and interfaces, you must capture the state behavior of the use case in a state diagram. Finally, in order to baseline the black box behavior of the use case, the state machine is executed and the generated sequence diagrams are compared against those created as scenarios earlier. The state machine of this use case is shown in Figure. 5.

### Figure 5. Black box state diagram



The state ‘Search Executed' has two ‘and' sub-states: ‘Perform Sensor Track Management' and ‘Perform History Check'. The first sub-state supports creating or updating tracks, while the second purges sensor track history greater than thirty minutes and the sensor track itself with no history.

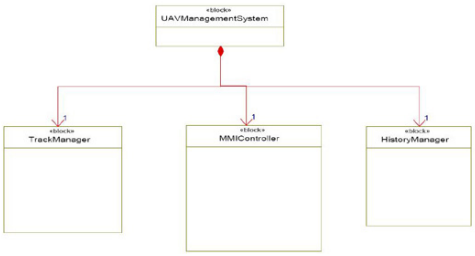
## Architectural design

In the architectural design phase, you need to emphasize structural decomposition and how operations and behaviors are allocated to the sub-system components. First, we describe the system BDD (see Figure 6) that structurally decomposes the system into sub-systems and then we shall arrive at the Use Case White-Box Activity Diagram to allocate the operations of the use case to the decomposed sub-systems (see Figure 7).

When the system is decomposed into sub-blocks, it is based on the definition of key system functions. The objective of this stage is to group the system functions together in a way that each group can be realized by a sub-system component. The first step is to group related system functions into key system functions. For the use case, we identified the following three key system functions through analysis of the use case black-box activity diagram:

- Manage Sensor Track
- Control Man Machine Interface
- Perform History Management

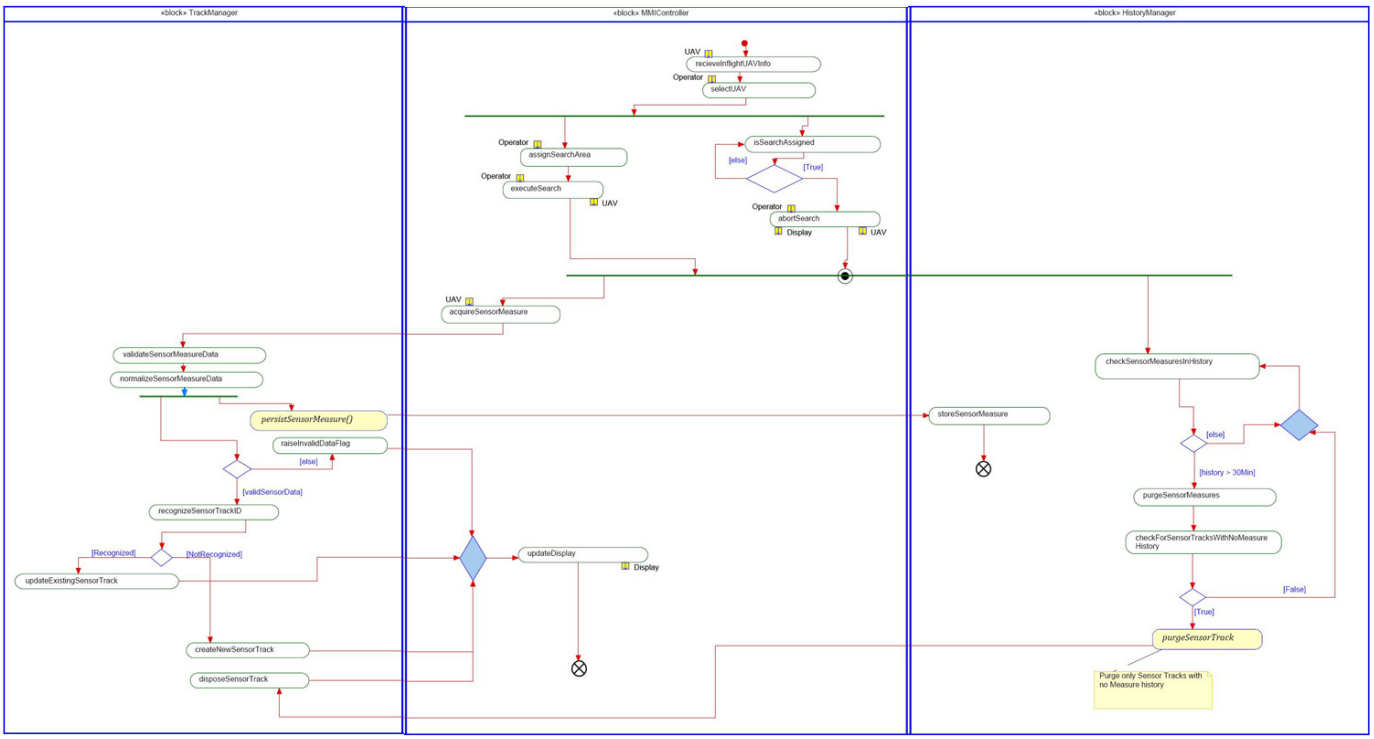
## Figure 6. UAV management system BDD




With the key system functions in mind, we derive the BDD as shown in figure 6. Since we have the subsystem blocks, the next task is to allocate operations across the swim-lanes to represent each independent sub-system block. The following rules are important for doing the allocation:

- If you cannot allocate an operation to a single block, then the operation must be decomposed. In this case, the associated decomposed operations must be linked to the parent operation through a respective dependency.
- You can allocate a system-level operation to more than one block. In this case, the relevant action is copied into the respective block swim lane and integrated into the functional flow.

## Figure 7. White box activity diagram



In figure 7, operations that involve an interaction with actors are clubbed in the Man machine interface (MMI) Controller component. Similarly the actions related to creating, updating and disposing of sensor tracks are allocated to the Track Manager swim lane. The actions related to historical data management are pushed to the History Manager swim lane. In places where the contiguous flow has been split into two blocks, message actions are used to indicate a forwarded request from one block to another. An example of this pattern is the purgeSensorTrack() message action from the History Manager component to the Track Manager component requesting the latter to disposeSensorTrack().

Now that the operations are allocated to the swim lanes, the next step is to perform the detailed architectural design.

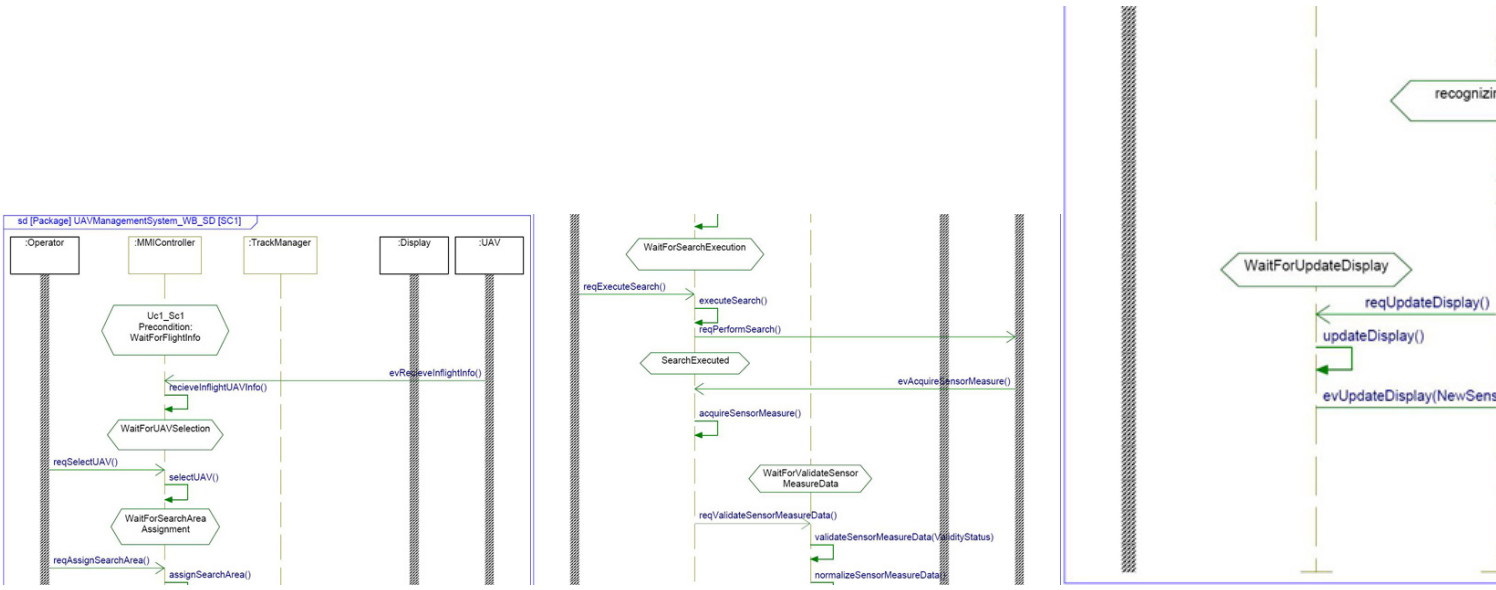
## Detailed architectural design

In the detailed architectural design phase you need to emphasize the definition of ports and interfaces, as well as arriving at the state-based behavior of the sub-system blocks. In order to do so you have to identify the sub-system block ports and interfaces by using the white box sequence diagrams..

While the focus of the black-box activity diagram was to identify the different flows of system functions (operations), the white-box activity diagram focuses on collaboration between the different sub-systems while taking the allocation of the operations into consideration. The service requests that are received define the interfaces of a block. After ports and interfaces are defined, the resulting state-based behavior of each leaf block must be captured in a state chart diagram.

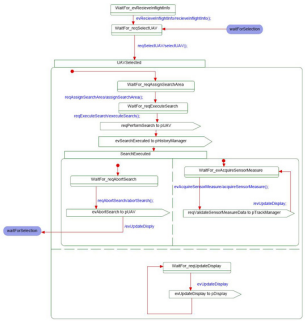
A delegate white box sequence diagram is shown in Figure 8. The sequence diagrams present services that are requested from one sub-system block to the other to satisfy the scenarios.

## Figure 8. White box sequence diagrams

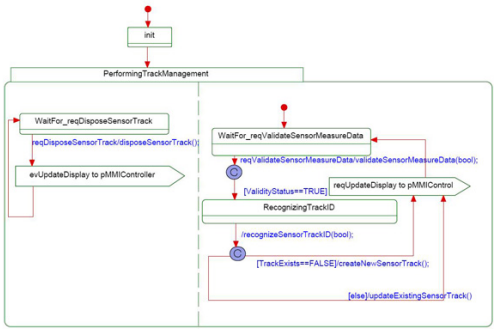



We continue to use the white box sequence diagrams to derive the ports and interfaces among the sub-systems and also to derive the state-based behavior of a delegate sub-system component, which is shown in Figures 9, 10 and 11.

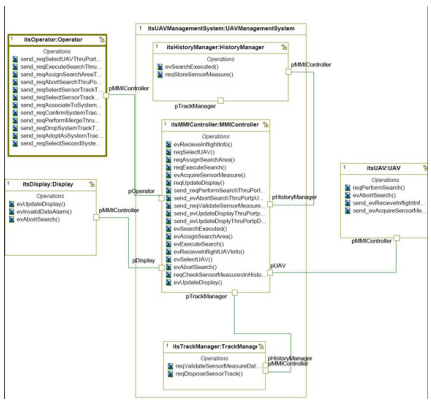
## Figure 9. MMI controller state



## Figure 10. Track manager state chart




## Figure 11. White box ports and interfaces



# Conclusion

We described how to apply the Rational Harmony systems engineering process though the uses of a case study. The key artifact of the hand-off from systems engineering to the subsequent system development is a baseline executable model. This model is the repository where you generate specification documents and operational ICDs. The hand-off packages contain the following items:

- Baselines of executable sub-system model(s)
- Definition of sub-system allocated operations
- Definition of sub-system ports and logical interfaces

- Definition of sub-system behavior, captured in a state chart diagram
- Test scenarios, derived from system-level use case scenarios