

Use Cases and Autonomous Characters

A technique for specifying tactical behaviour for models of military decision makers

Michael Papasimeon and Clinton Heinze

Air Operations Division

Aeronautical and Maritime Research Laboratory

Modelling for Simulation

- Simulation as an Operations Research tool
- Modelling the human component
 - Interviews with subject matter experts
 - Standard procedures
 - Manuals
 - Observation
- How do we *manage* this knowledge?
 - In general?
 - For building software?

The Source Code **was** the model

- ... and the knowledge base,
- and the configuration management system,
- and the design documentation,
- and the test harness,
- and the *only* artefact of the software engineering process.

But it should not be any of these things!

It should be just the source code!

Continuous Improvement

- SWARMM
 - Requirements specifications
 - Architectural and detail design documents
 - Tactical behaviour documents
- BattleModel
 - Object Orientation
 - UML
 - Process

Tactical Behaviour Lagging

- Everything else strongly OO - good tool support
- AOD agent experts not software engineers
 - See the problem but not the solution
- Source code (dMARS Plans) still formed an *integral* part of the model
- Need to improve the software engineering of the models of tactical behaviour

Our Problem

- Need to specify tactical behaviour of real life military decision makers (pilots, fighter controllers, radar operators)
- Need clear, concise, unambiguous, and complete specifications (as for all software)
- Specifications may be used as a basis for design and implementation of a software model
- Want the specification of behavior independent of the implementation choice

Possible Solutions

- Traditional Requirements Specification
- Textual Descriptions
- Formal Specifications
- Flow Charts and Activity Diagrams
- Use Cases
- Story Boards
- Finite State Machines

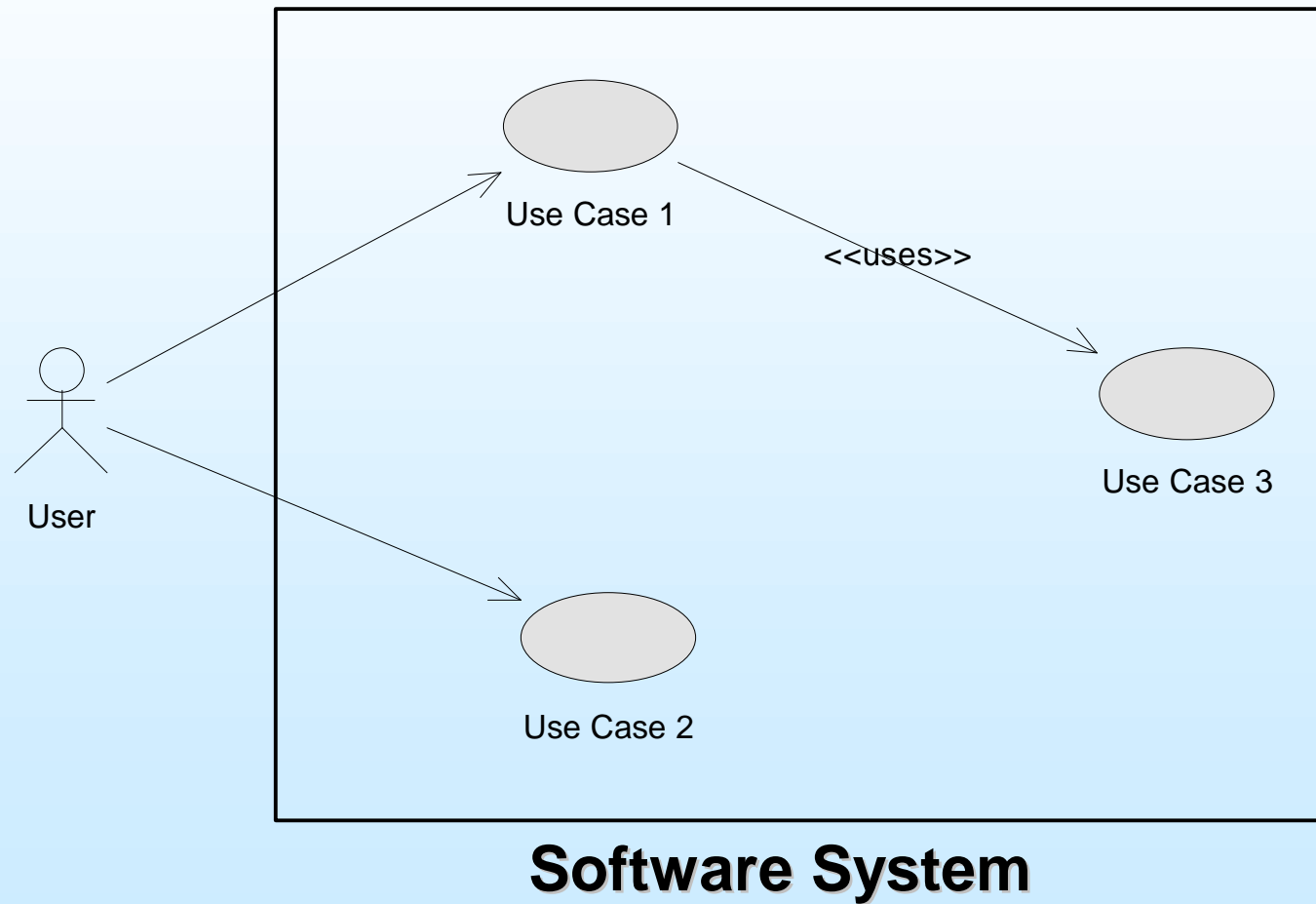
Use Cases - What are they?

- A technique for organising functional requirements
- Specify the behaviour of the system from the perspective of external entities known as actors (users, software/hardware systems)
- Each use case provides some value to the actor initiating the use case or another actor

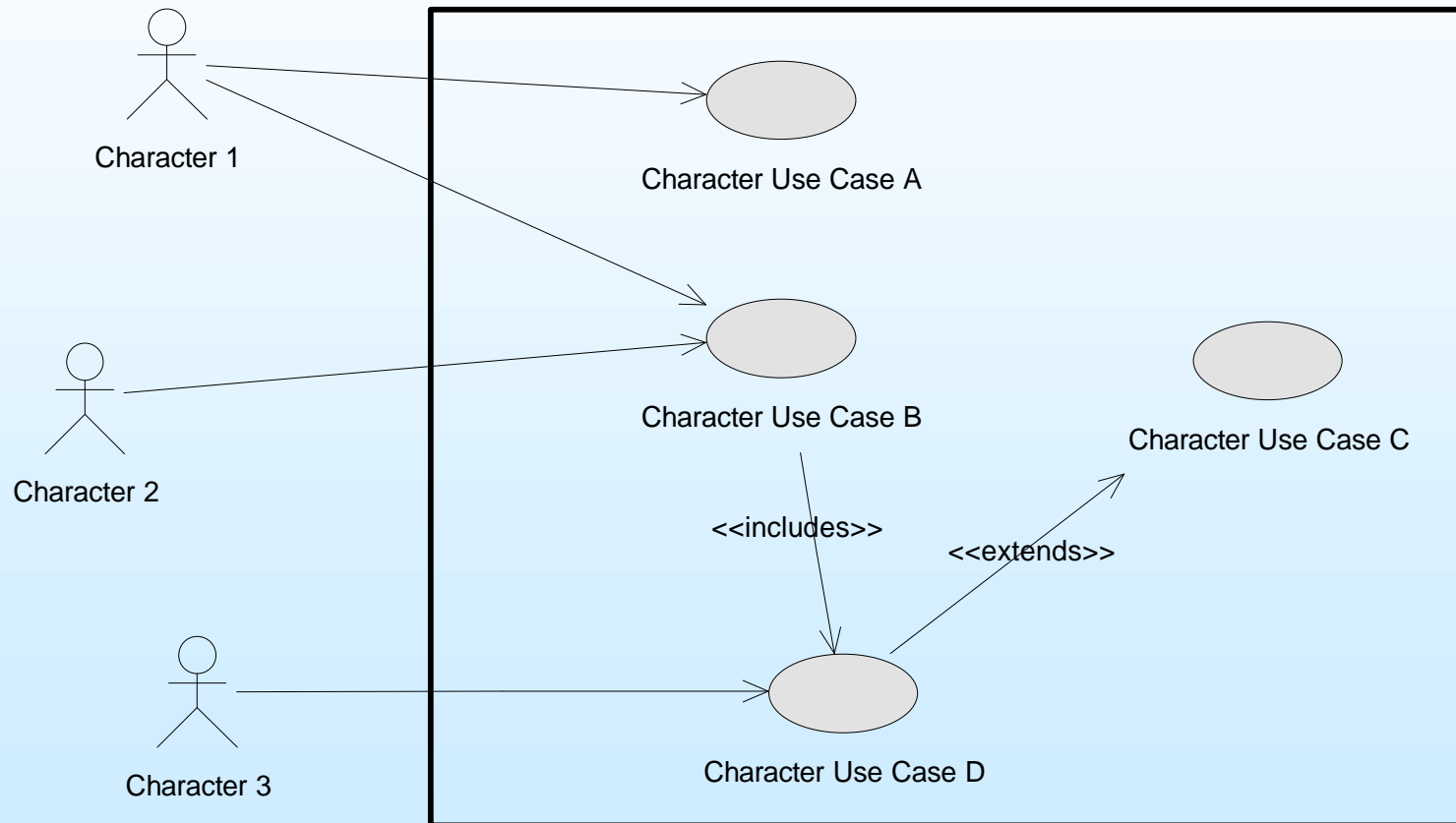
Use Case Documentation

- Descriptive Text
- Actor and Object Interactions
- Use Case Associations
- Pre Conditions, Post Conditions
- Flow of Events, Alternative Flows
- Sequence, Collaboration and Activity Diagrams

An Interactive Software System

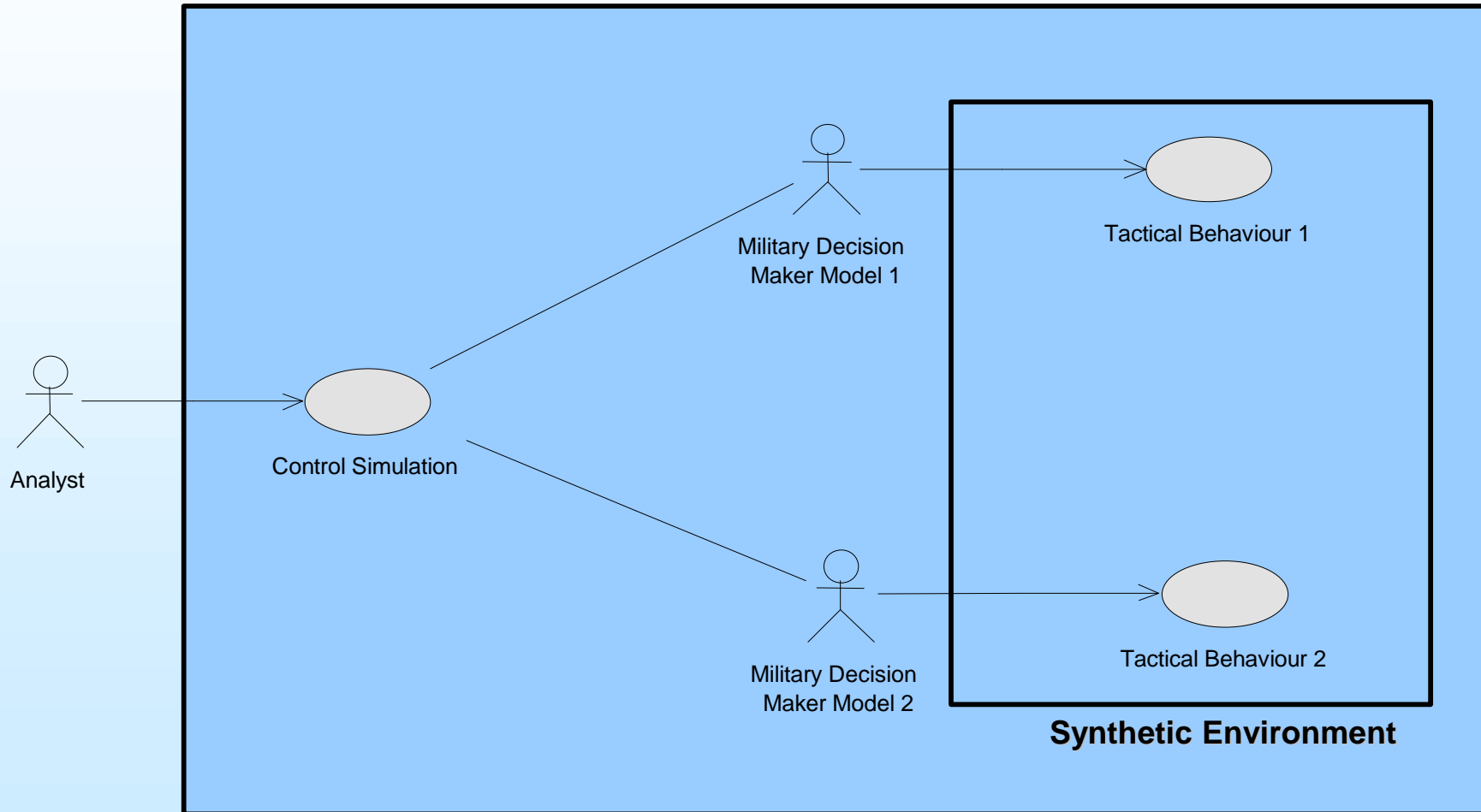


Characters Interacting with a Synthetic Environment

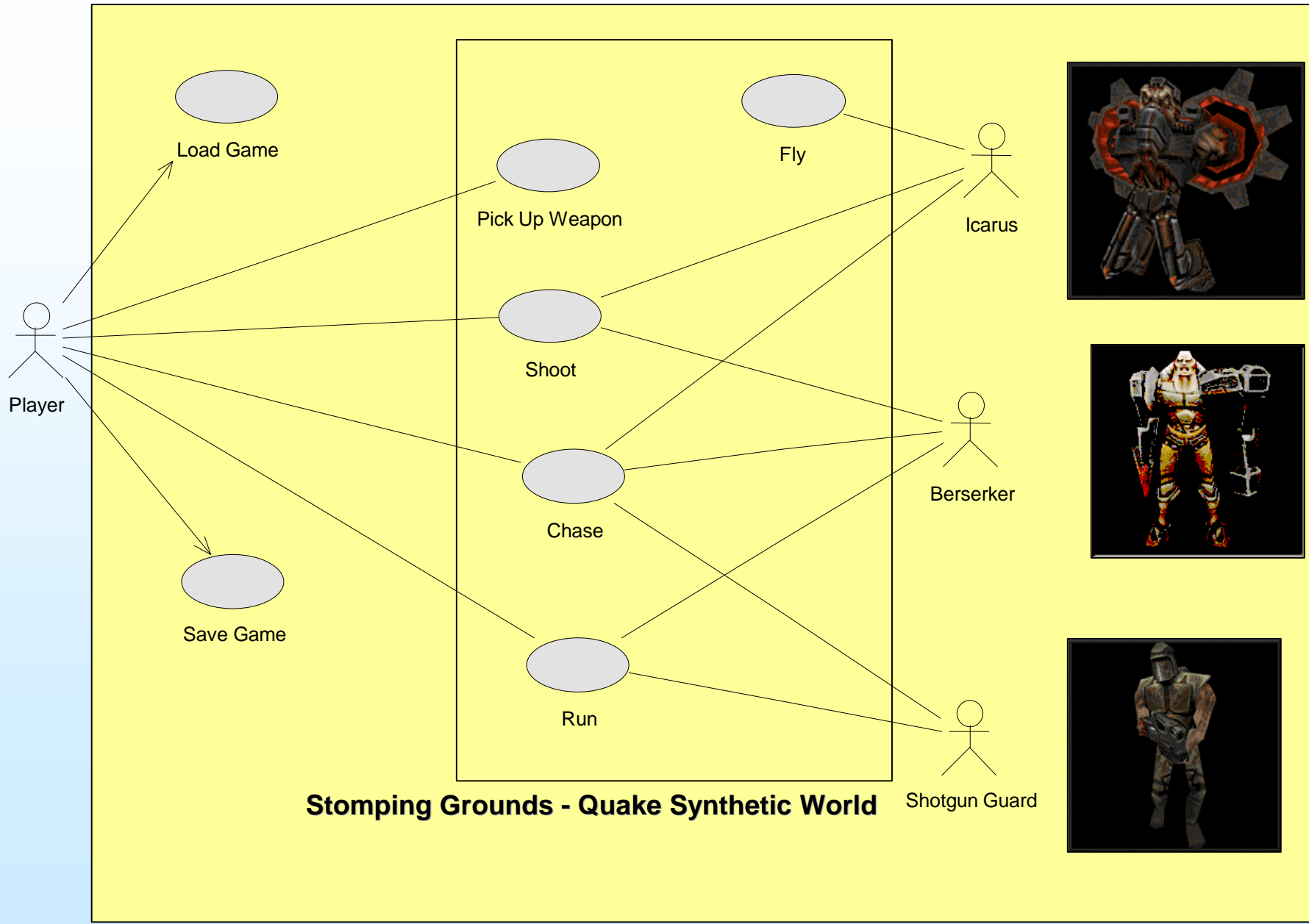


Synthetic Environment

The Systems We Build



Software System - Military Simulation



Stomping Grounds - Quake Synthetic World

Quake II

Specifying a Use Case Model

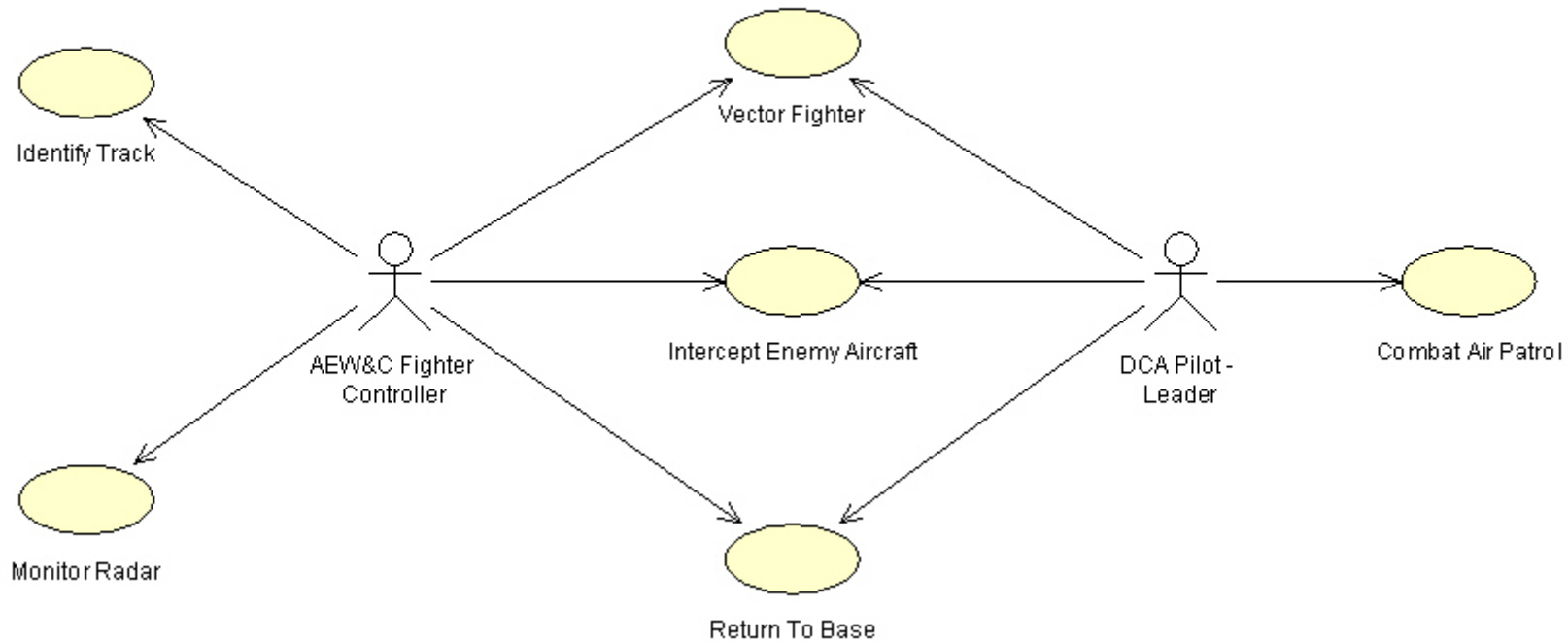
- Define Scenarios
- Identify Actors
- Identify Use Cases and Relationships
- Document Each Use Case
 - Pre-Conditions, post-conditions, event flow, alternative flows
 - Identify objects participating in use cases (sequence and collaboration diagrams)
 - Activity and state diagrams (graphical tactical specification)

Case Study 1 - Bonestorm

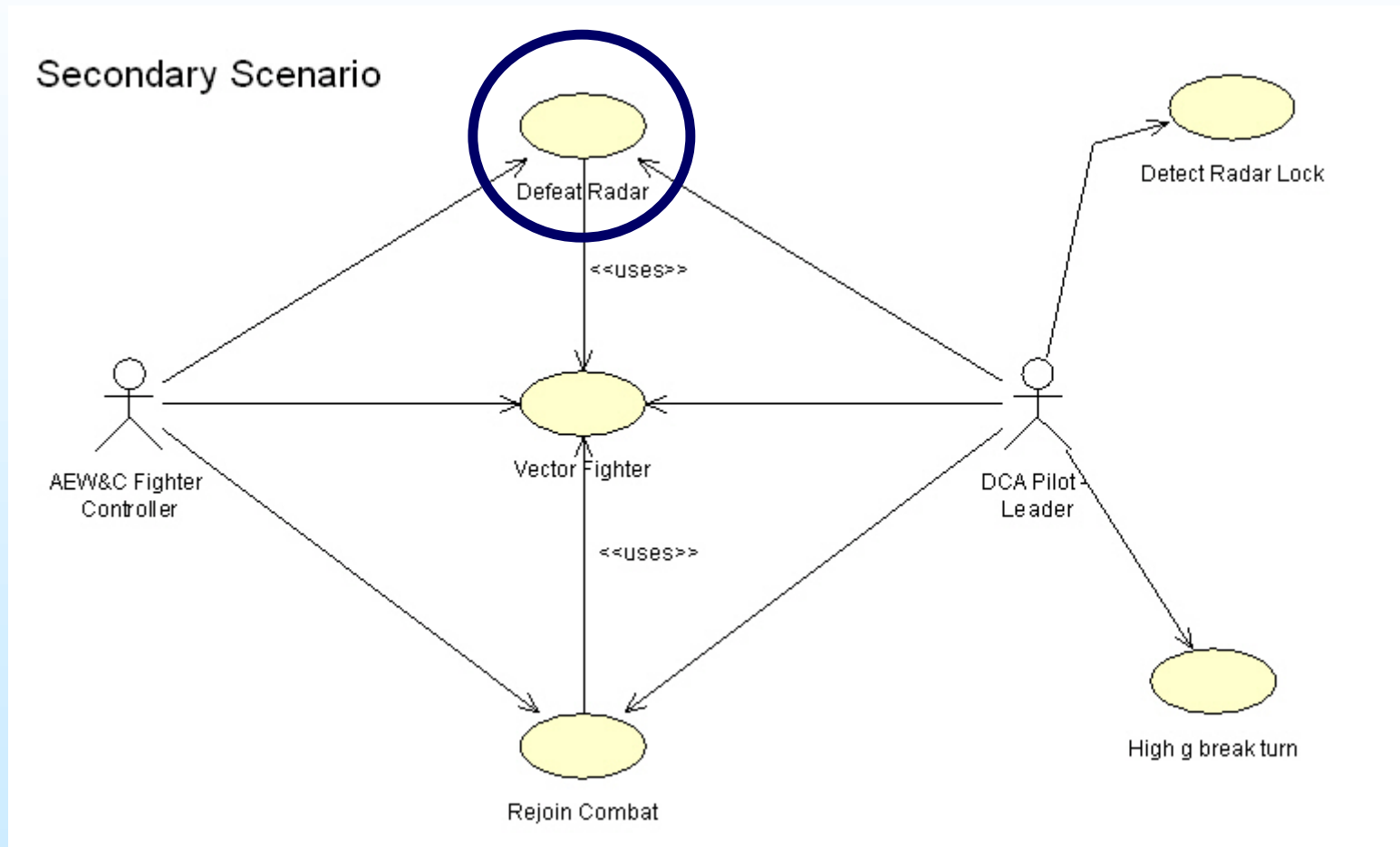
- A user interface for a Fighter Controller
- Fighter Controller:
 - is on the ground
 - or in the air on AEW&C
 - issuing commands to fighter pilots in combat zone.
- Commands
 - Scramble
 - Vector
 - Return to Base (RTB)

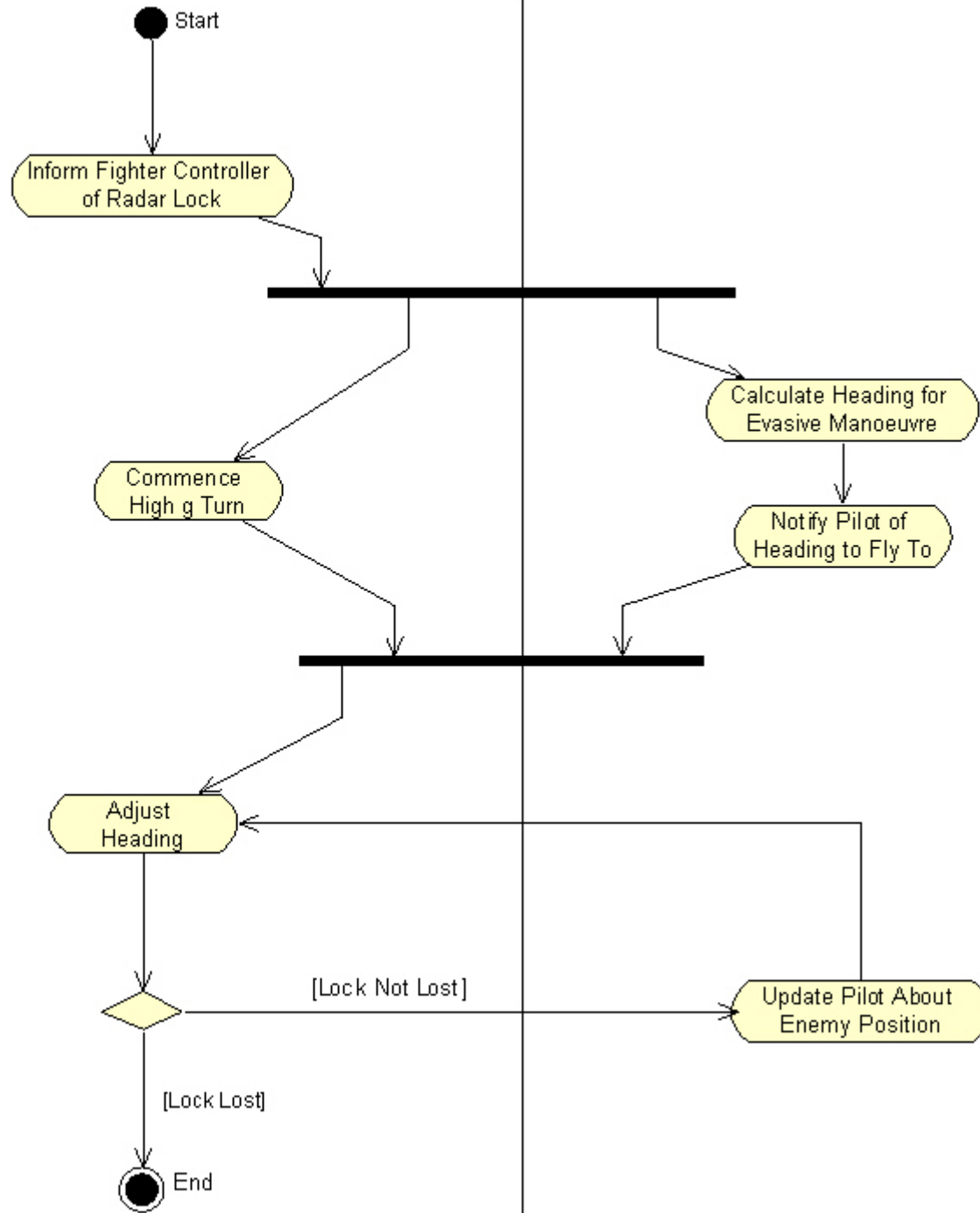
Fighter Controller Interface Use Case Model

Primary Scenario

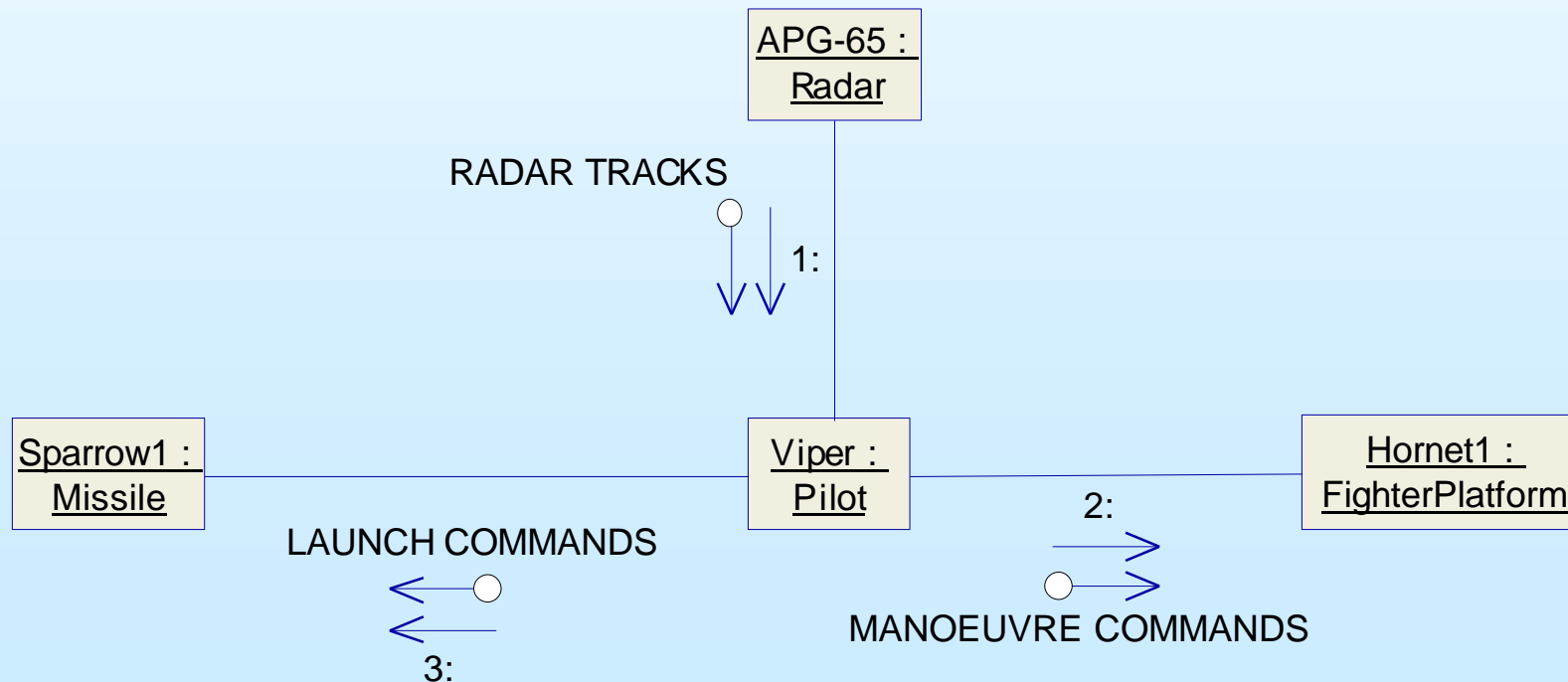


Use Case Model (2)

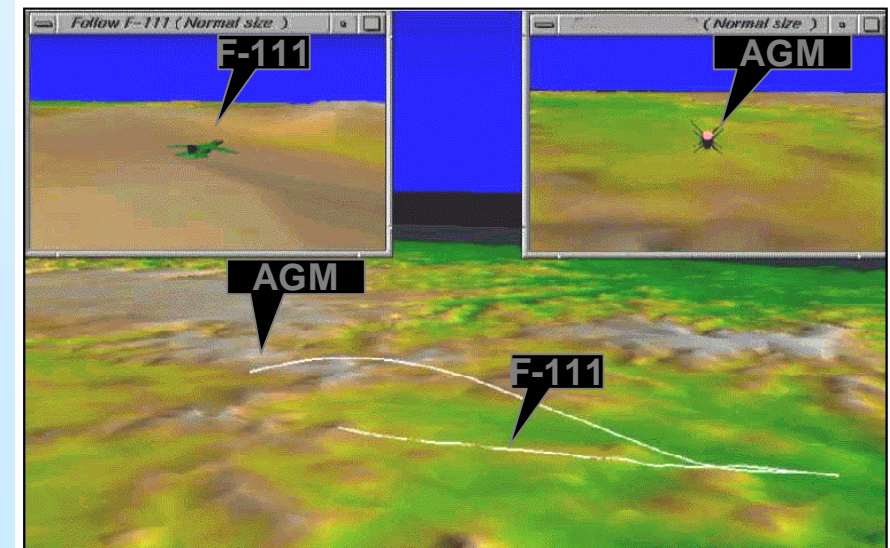
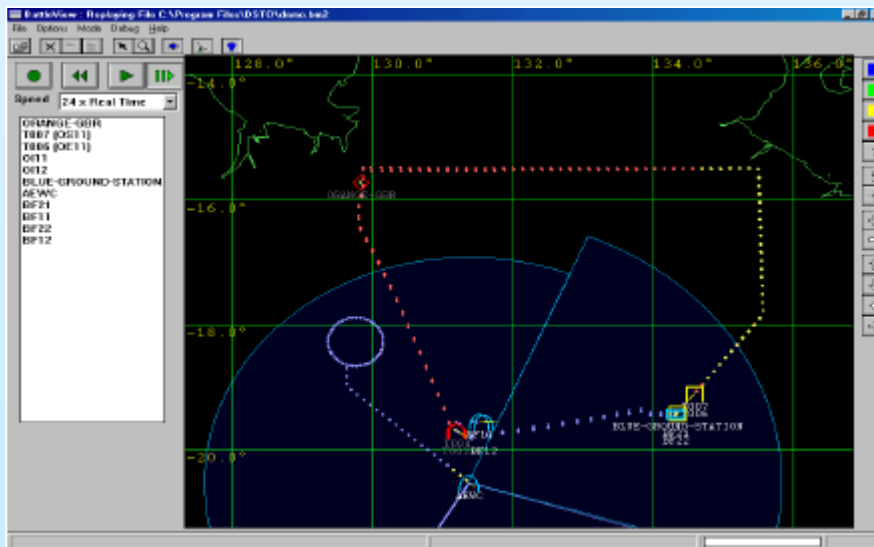
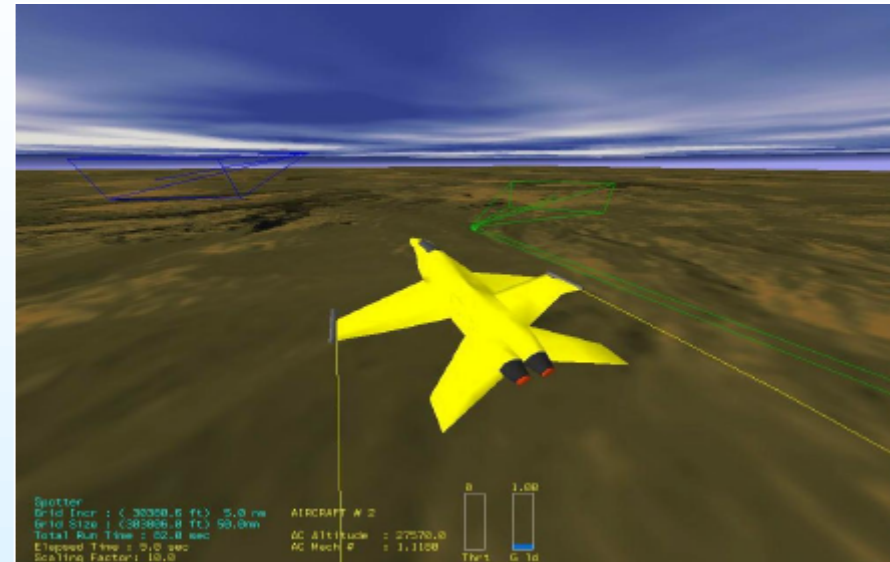
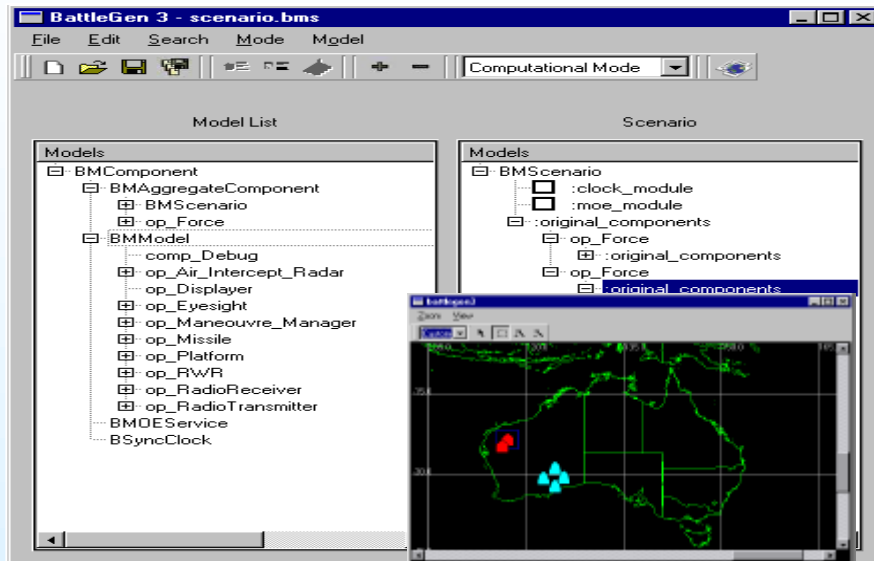




F/A-18 Hornet Component Interactions

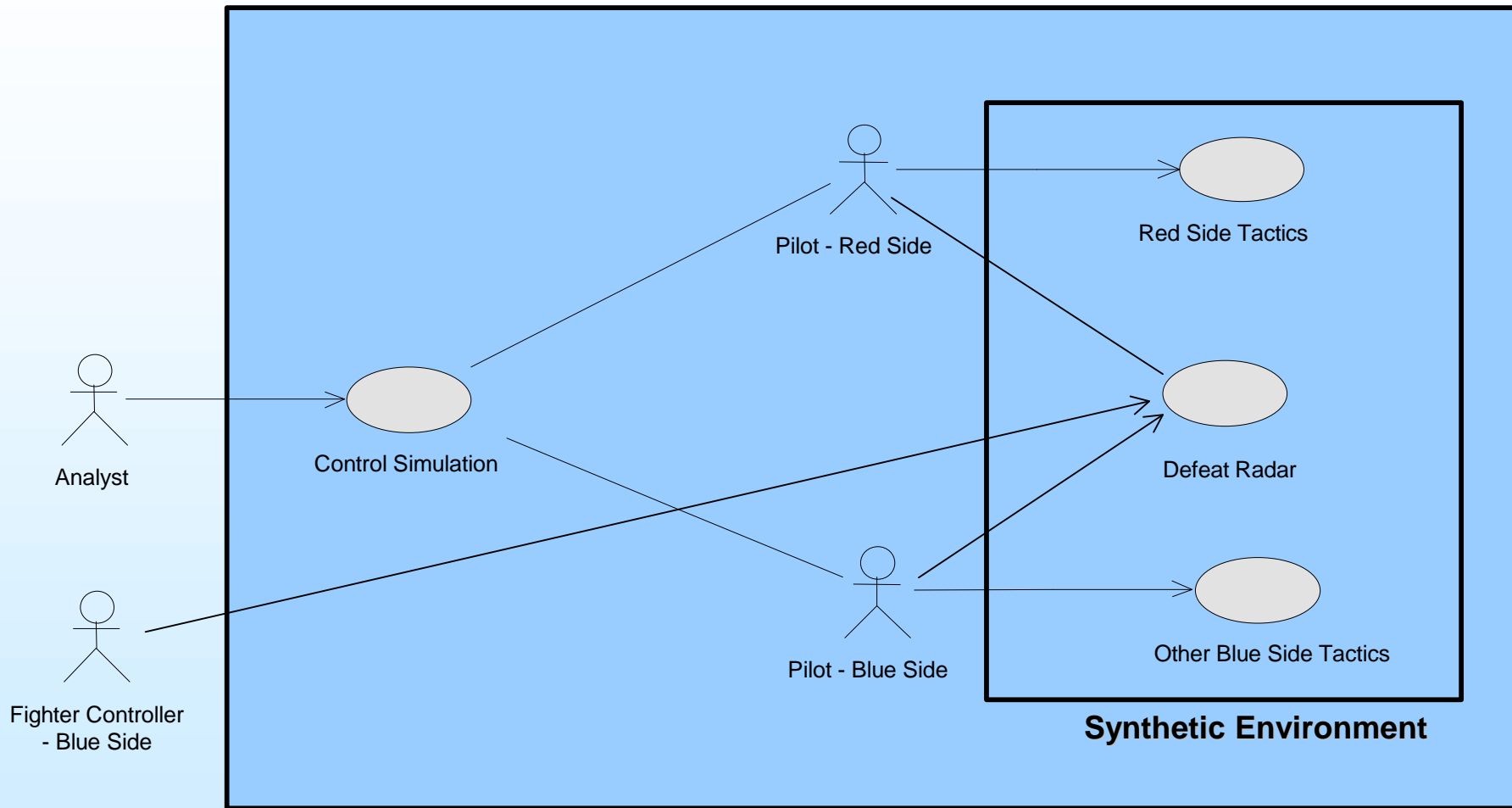


Example User Interfaces for GCI

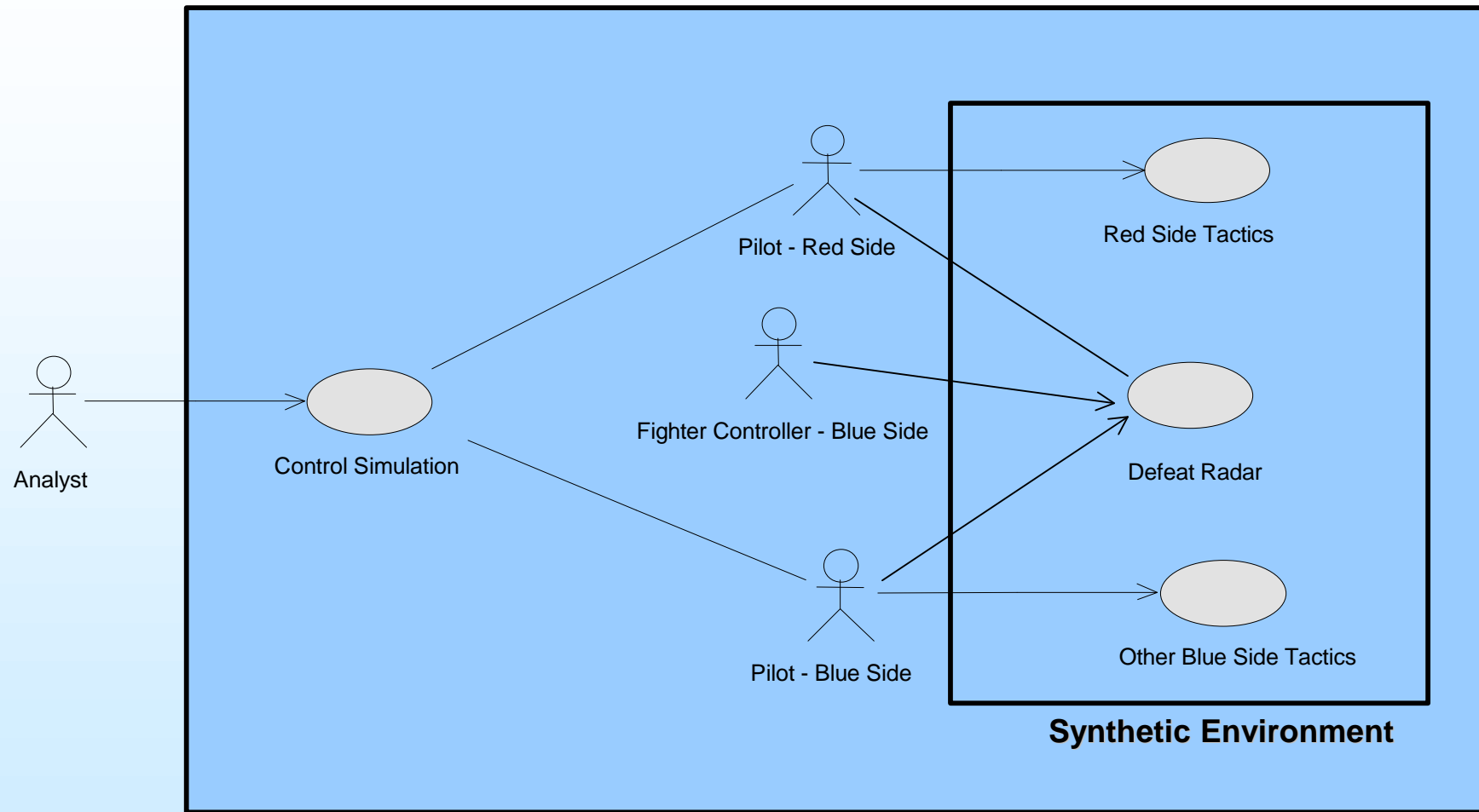


The same system for Operations Research

- Large numbers of runs, many parameters varied and studied
- Replace human fighter controller and GUI with a computational model
- Must specify the required tactical behaviour of that model



Bonestorm - Human-in-the-loop



Bonestorm - Constructive Simulation

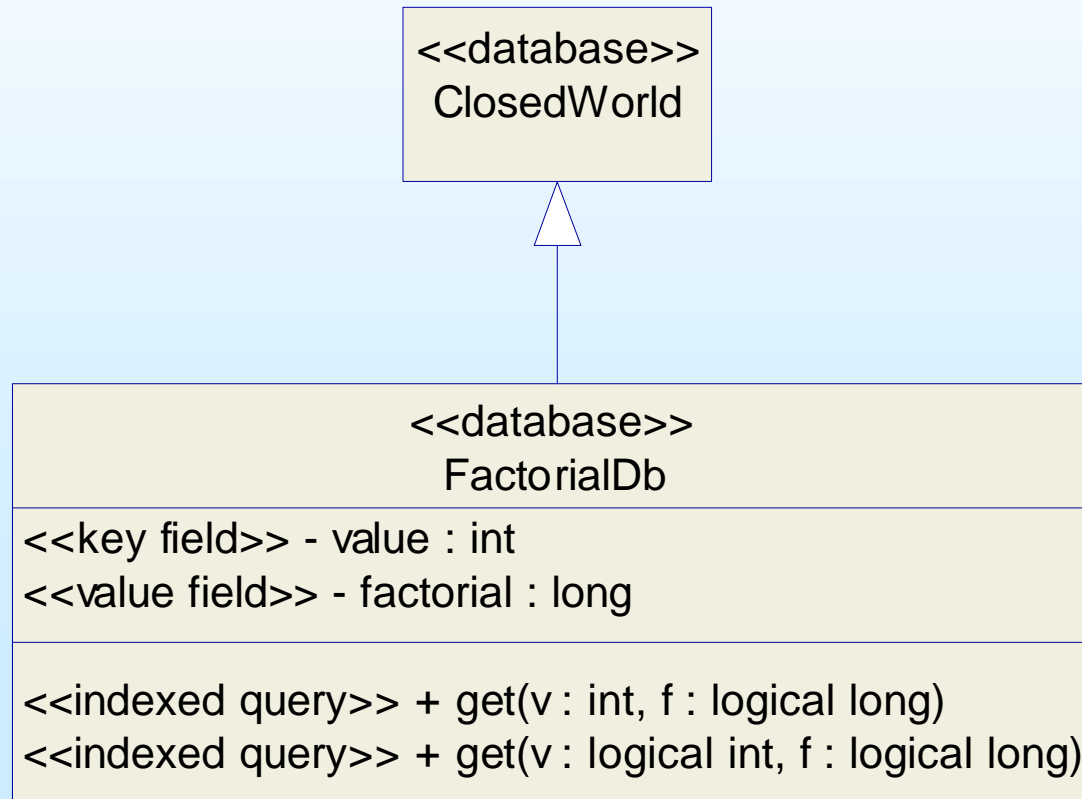
UML Extensions for JACK

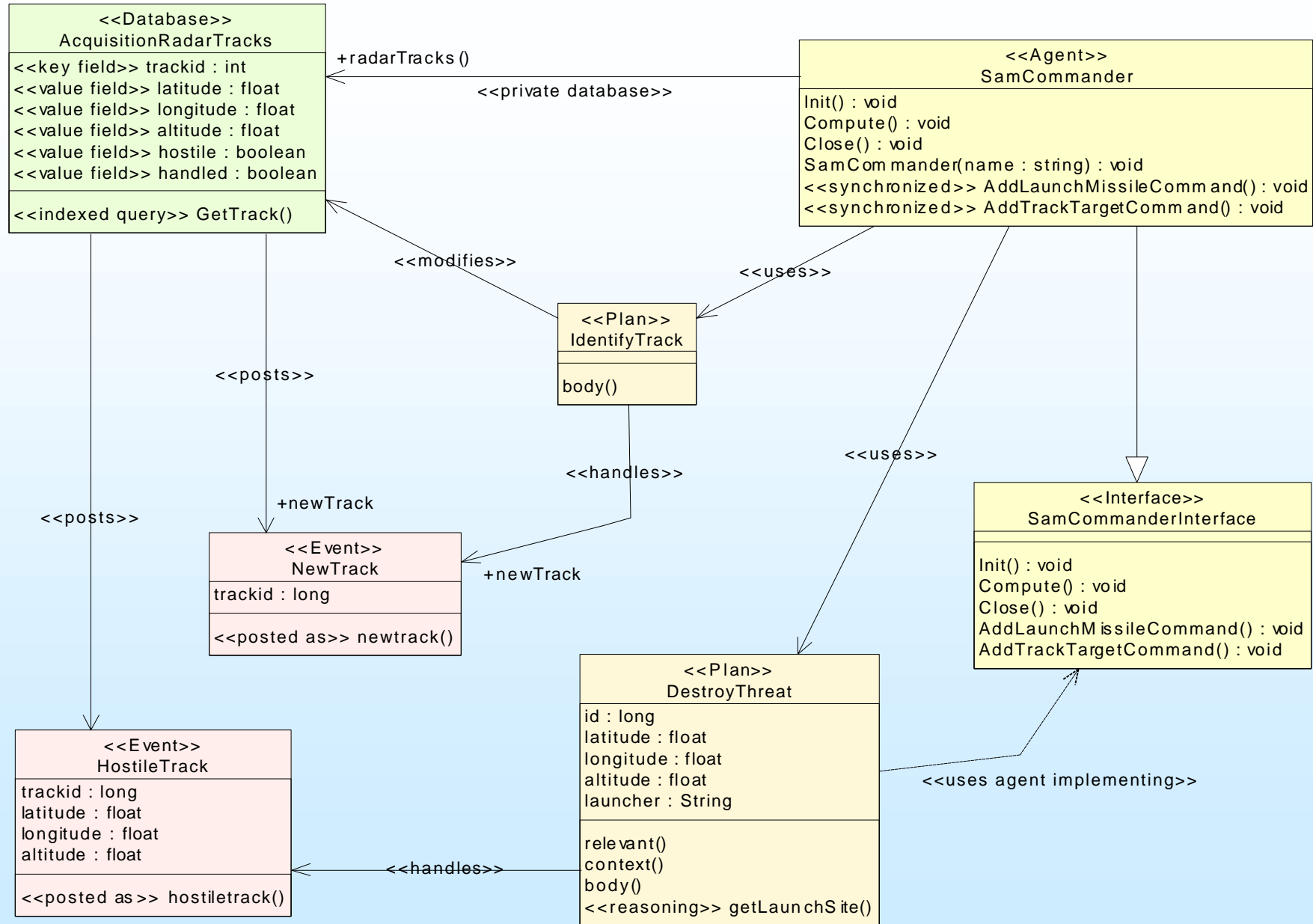
- JACK = Java + Extensions + Agent Architecture
- UML can easily be used to model Java systems
- UML can be extended to support JACK at the detailed design and implementation level using <<stereotypes>>

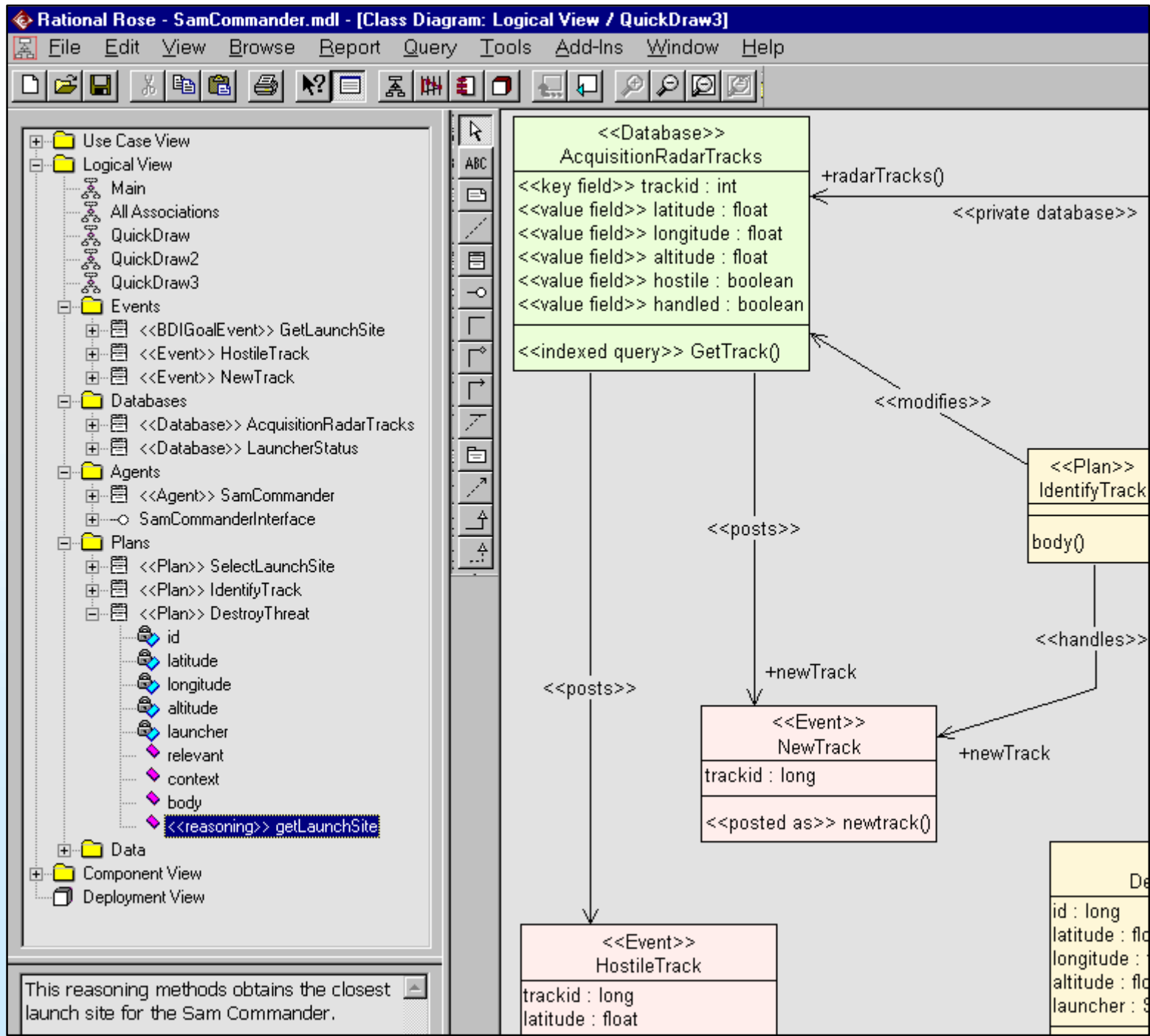
JACK Agents - Detailed Design

- High Level JACK Stereotypes
 - <<agent>>, <<plan>>, <<event>>, <<database>>, <<capability>>
- Association Stereotypes
 - <<uses plan>>, <<handles event>>, <<posts>>, <<reads database>>
- Low Level Stereotypes
 - <<key field>>, <<value field>>, <<indexed query>>

```
database FactorialDb extends ClosedWorld {  
  #key field int value;  
  #value field long factorial;  
  
  #indexed query get(int v, logical long f);  
  #indexed query get(logical int v, logical long f);  
}
```







UML Packages and JACK Capabilities

