A Model Problem Approach to Measurement-to-Track Association

Grace A. Lewis
B. Craig Meyers

September 2003

TECHNICAL REPORT
CMU/SEI-2003-TR-020
ESC-TR-2003-020

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

A Model Problem Approach to Measurement-to-Track Association

CMU/SEI-2003-TR-020
ESC-TR-2003-020

Grace A. Lewis
B. Craig Meyers

September 2003

COTS-Based Systems Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Abstract

This is the first in a series of reports that illustrate the use of model problems in the design of a system. The problem considered is measurement-to-track association. A “track” represents the state data about an object in the environment, and has a set of associated attributes. “Measurement-to-track association” is the process of determining the relation between a measurement and an existing track. In this process, tracks that meet particular attribute criteria can be selected via filters. This report examines the development and application of filters that can be used as selector mechanisms. The report also presents an initial design of the model problem, by using concepts and constructs from Unified Modeling Language (UML), Executable UML (xUML), and Object-Oriented Analysis (OOA). Also covered are possible extensions to this work, related to performance considerations, additional filter types, and the distribution of filter information.

1 Introduction

There are many issues in the development of large, complex, distributed systems. There are just as many approaches to dealing with these problems. In this report we start to develop a model problem that is representative of an important problem for a particular class of systems. The problem considered is measurement-to-track association and the development (and application) of filters that can be used as a selector mechanism. These filters must be sufficiently general that they can be constructed, and applied to, an arbitrary class/object combination.

This is the first in a series of reports that illustrate the use of model problems in the design of a system. This report focuses on the specification of the problem and an initial design. The choice of a final design is influenced by many factors. In this case, we are especially interested in performance properties of the system. Hence, subsequent work will illustrate the use of a qualitative performance model, as well as quantitative aspects of that model to yield a solution. Performance considerations can often drive a solution approach and there are iterations between a design and information concerning the performance of that design.

This report is organized in the following manner: In Section 2 we describe the role of model problems in general, and then specialize that discussion to the problem at hand. A Unified Modeling Language representation of the problem appears in Section 3. Possible extensions of this work can be found in Section 4. A brief summary of the report appears in Section 5. A number of appendices accompany this report which describe details associated with use cases, classes, sequence diagrams, and a potential alternate design for the model problem.

We acknowledge discussions with Holly Hamilton and Brad Leon during the development of this report.

2 Model Problem

2.1 Model Problem Selection Criteria

It is appropriate to briefly describe how we use the term *model problem*. In assessing the selection of a model problem we are concerned with several factors, among them

- *Is the model problem common to the design of a system?* Although there are many aspects of the design of a system, there are cases where such aspects have a recurring theme. Because such themes may appear in many aspects of the solution, the more common they are, the more likely a solution to them will have value. That is, one hopes that the solution to a particular model problem can be reused across the design of the system.
- *Does the model problem help mitigate a risk?* There are many potential risks associated with the design of a system. We are concerned with problems that present a risk whose consequences could adversely affect some aspect of system operation. In particular, the aspects we are concerned with are those related to system performance, reliability, or other quality attributes.
- *Does the solution of the model problem lend itself to reuse in other contexts?* For example, suppose one wishes to develop a performance model of a system. If it is possible to develop a performance characterization of the model problem, then the understanding of that performance characterization may be applied to other contexts as well.

Various characteristic problems inherent in a system may exhibit different aspects, related to the above. Of course, when a model is characterized by multiple characteristics, it assumes even more importance to the successful development of a system solution.[1]

2.2 Model Problem Definition

The overall context for the model problem treated in this report is track management. We define a track to represent the state data about an object in the environment. A track has a set

1. Note that in no way do we use the term *model problem* to be synonymous with a *toy problem*. Toy problems are used for discovery or familiarization and are usually meant to be thrown away. Model problems, on the other hand, are focused on solving a particular problem and the results are documented to guide design and implementation.

of associated characteristics, or attributes. For example, an aircraft might have position and speed as attributes. The context for the overall problem appears in Figure 1.

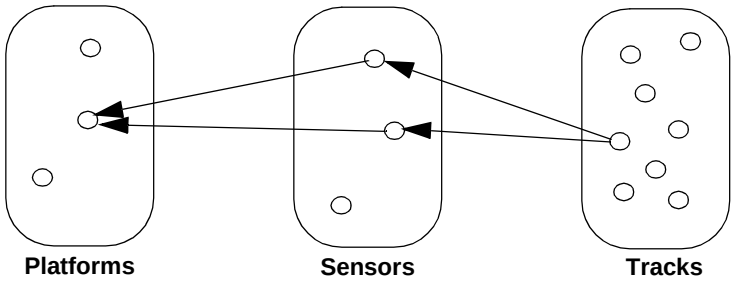

Figure 1: Overall Context for Model Problem

The processing suggested in Figure 1 includes the following:

- There is a set of tracks in the environment.
- Sensors provide measurements about tracks. Sensors may be of different types.
- Platforms contain one or many sensors.

We do not seek to identify all the details associated with the basic processing shown in Figure 1. For example, a significant problem is the distribution of information (such as track data or sensor measurements) among multiple platforms. There clearly is a difference between a platform-centric view of the environment and a multi-platform view of the environment!

An interesting aspect of track management deals with association as defined below:

> ***association:*** *the process of determining the relation between a measurement and an existing track.*

We will call this type of association *measurement-to-track association*. During measurement-to-track association, initially there is a set of tracks. At a time *t* a sensor performs a measurement of the environment which results in performing the following operational thread:[2]

1. **[Input]** The data collected by the sensor is reported to the system through a communications link. The sensor may report values such as the latitude and longitude of contact information.

2. This operational thread assumes the existence of tracks that have been created over time as a result of the received measurements. At system initialization there are no tracks.

2. **[Validation and Pre-Processing]** The reported data is first validated. For example, there may be acceptable ranges of data and these are checked to make sure that the reported data is not corrupted. Then the data has to be adjusted to account for any bias in the sensor itself.

3. **[Distribution]** The sensor measurement data may be distributed to other systems, some of which may reside on different platforms than the platform providing the sensor measurement.

4. **[Transformation]** The measurement data may have to be transformed so that it is consistent with track data. For example, some form of coordinate conversion may have to be applied to the measurement data.

5. **[Filtering]** From the set of all tracks, a subset of tracks is chosen as a candidate match for the measurements. For example, the criteria for filtering tracks may be based on nearest-neighbors: Only those tracks that are within a certain distance of the position of the measurement are considered. This aspect involves development and application of a *filter*. Note that the filter is created dynamically (during runtime) based on the position of the measurement data reported.

6. **[Propagation]** The candidate tracks have associated state data that is valid at times other than the reported measurement time *t*. Hence, the candidate tracks are propagated in time so that the track data is extrapolated to the time of the measurement data.

7. **[Re-Filtering]** The propagated tracks have to be re-filtered to make sure that they are still within the initial filtering criteria; it is possible that an initial candidate track is propagated out of the filter.

8. **[Evaluation]** Algorithms are applied to the propagated track data and the reported measurement to determine the likelihood that the measurement should be associated with an existing track. Comparison of the appropriate track attributes are made between the candidate tracks and the attributes provided by the measurement. The evaluation algorithms may vary depending upon the type of measurement.

9. **[Decision]** Based upon some selection criteria it is determined if the measurement report should be associated with one of the candidate tracks. Several options are possible:

    - If any of the filtering steps returns no tracks, or if all candidates tracks have an association value that is below some acceptable threshold, an algorithm is applied. This determines if the measurement provides additional data so that one can decide, from looking at the set of yet unassociated measurements, if there is enough information for a new track to be created. If there is enough information, a new track is created based on the reported measurement data plus the set of related unassociated measurements.
    - For the candidate track whose association value is larger than some acceptable threshold, track data is updated with the reported measurement data and the measurement is associated with that track.[3]

The above appears simple and straightforward, but it is not; for example, the algorithm to evaluate measurements against tracks is non-trivial. Further, there are performance implications for the end-to-end processing of a measurement.

2.3 Model Problem Selection

The selected model problem is the association of a measurement report from a sensor to a track—*measurement-to-track association*. The model problem design and implementation will be presented in the context of a specific modeling and code generation tool.

Measurement-to-track association as described in Section 2.2 raises a number of issues.

- Filtering and Re-Filtering: There are several questions regarding filtering that must be answered as part of the measurement-to-track association process. For example, what is returned by the application of a filter when it produces a set of filtered tracks? Possibilities include that
  - a list of the identifiers of the tracks satisfying the filter criteria is returned
  - a list with copies or "clones" of the matching tracks is returned

  There are also general questions about the character of the filter that is applied. A filter could represent a general query corresponding to certain values of the track attributes. It is a general query in that the filter can be applied to attributes on a one-to-one basis. However, other forms of a query are possible. For example, if one wishes to construct a filter that can be applied to determine a candidate set of tracks within a specified range of a given point, such a filter would be based on an algorithm whose parameters are attributes of the object. In the case of the range, the algorithm would involve a computation of distance. We can also envision cases where a composition of filters may be applied.
- Evaluation: The algorithms for evaluation are very complex and computation-intensive. What are the performance consequences of this intensive and complex computation that is performed many times?
- Data structure design: It is common for tracking systems to use specialized structures and algorithms for the storage and manipulation of data such as track data. For example, hash coding schemes and algorithms are often used as a way to increase search performance. However, if one wished to adhere to a purely object-oriented approach and the constructs of a chosen implementation language, what are the implications for data storage and manipulation?

3. We assume that a measurement associates to only one track.

Given these issues and recognizing the need for an end-to-end problem in the context of a single platform, it is worth assessing the combination of filtering, propagation, evaluation, and decision as a model problem in light of the criteria specified in Section 2.1. In particular,

- The use of data filtering and propagation is very *common* to the system under consideration.
- There is a *risk* associated with the use of filters as described in the tentative design. In particular, if the time to create and apply a filter to a system that contains a large number of tracks is prohibitive, then it may warrant consideration of a different design approach. The complexity of the algorithms for evaluation (and some of the decision making) could also require large amounts of computation. Furthermore, because of the amount of measurement data that is received, the association process is performed many, many times.
- If there is performance data about the elements of the design (such as the time to create a filter or to perform propagation and evaluation) then that information may be integrated to yield a performance model of a thread of the system (in this case, we define a thread to be the sequence of operations described in Section 2.2 on page 3). Thus, we can reuse the solution approach to the model problem in the context of a *performance model.*

The validation, pre-processing, distribution, and transformation aspects of measurement-to-track association will not be considered part of this model problem. Validation, pre-processing, and transformation are relatively straightforward and should be implemented outside of measurement-to-track association so that only valid measurements are provided for association. Distribution is a different problem of sufficient importance that it should be treated in a general context. In particular, one is concerned with the manner in which state data about the system is distributed among its constituent elements.

Taken together, the preceding discussion illustrates the value of model problems to the design of a system. Model problems can help answer important questions such as: If there is a system with a very large number of objects, what are the performance implications for a given design approach? Is each measurement-to-track association operation going to be a separate thread? What is the time to process a measurement vs. the arrival rates of measurements? What if data starts coming in faster than the thread can process? What is the real end-to-end performance of the system? Can all these questions be addressed in the context of the selected modeling and code generation tool? All these questions must be addressed in order to develop a successful implementation.

3 Modeling of the Measurement-to-Track Association Model Problem Using UML

3.1 Approach

In this section we illustrate the measurement-to-track association model problem using Unified Modeling Language (UML) and concepts from Executable UML (xUML) [Mellor 02] and Object-Oriented Analysis (OOA) [Shlaer 92].

Briefly, the process used involves the

- identification of use cases related to measurement-to-track association
- partitioning of the measurement-to-track association problem into domains
- development of sequence diagrams corresponding to the domain interaction to satisfy the execution paths listed in the use case descriptions
- development of a class diagram for each domain
- development of sequence diagrams corresponding to the object interaction to satisfy the execution paths listed in the use case descriptions
- development of a class collaboration diagram for each domain

At this level of analysis we do not address concurrency issues. These issues will be addressed during detailed design.

The main modeling tool used in this report is iUML by Kennedy Carter, Ltd. (http://www.kc.com). Rational Rose by IBM (http://www.rational.com/products/rose/index.jsp) is used for some of the modeling that is not supported by the iUML tool.

3.2 Use Case Diagrams

A use case represents a coherent unit of functionality provided by a system, a subsystem, or a class. A *use case diagram* shows the relationship among use cases within a system and its actors. *Actors* are external entities (people or other systems) who interact with the system to achieve a desired goal .

The use cases related to measurement-to-track association are represented in the use case diagram in Figure 2.

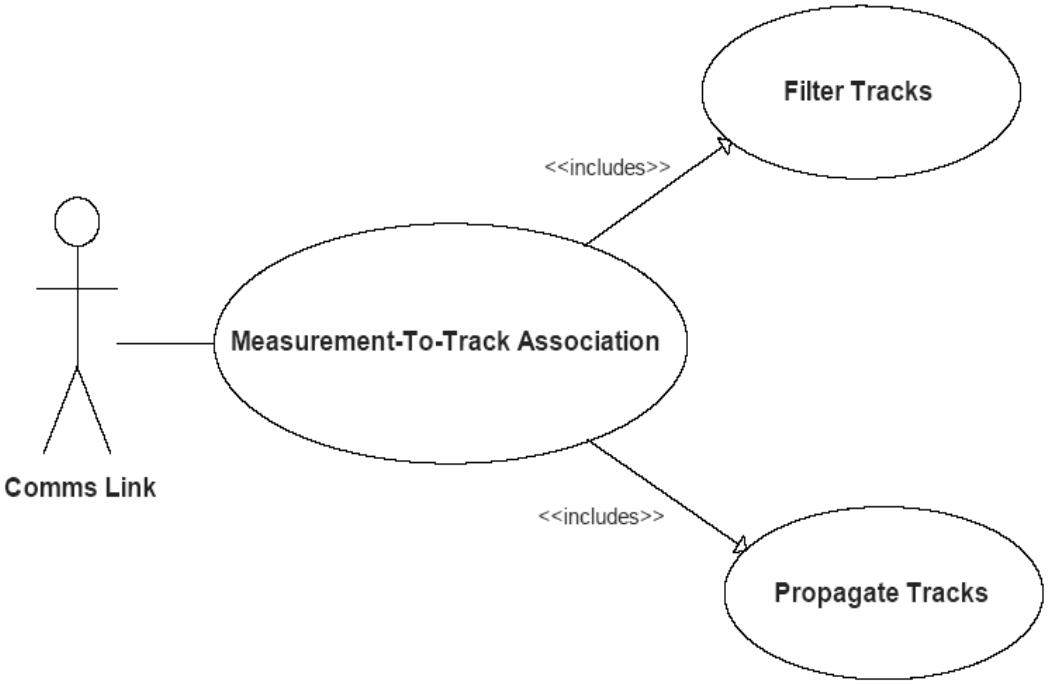

Figure 2: Measurement-to-Track Association Use Case Diagram

The main use case in the diagram is *Measurement-to-Track Association*. The actor that interacts directly with this use case is the communications link. Sensors communicate with the system through the communications link to provide measurements to be potentially associated with a track. The *Measurement-to-Track Association* use case includes two additional use cases: *Filter Tracks* and *Propagate Tracks*. The *Filter Tracks* use case searches a set of tracks for those matching the set of criteria provided by the filter and the *Propagate Tracks* use case propagates the characteristics (e.g. velocity, position) of a set of given tracks in time. These two use cases have been specified separately because we believe that they are of sufficient generality that they might apply to other use cases outside of measurement-to-track association.

There are a number of generally accepted formats for use case description [Booch 99, Cockburn 00]. The elements we will use are

- name: name of use case
- purpose: brief description of the purpose of the use case
- precondition: conditions that must exist before the use case takes place

- requirements satisfied: requirements satisfied by the use case (usually requirement number from requirements document)
- basic course: captures normal behavior associated with the use case
- alternate courses: capture unusual behavior such as exception handling and error behavior
- includes: use cases included by this use case
- included by: use cases that include this use case
- extends: use cases extended by this use case
- extended by: use cases that extend this use case
- communicates with: external entities (actors) participating in the use case
- performance specifications (optional)
  - *trigger*: external unsolicited event that initiates execution of the use case
  - *periodicity*: nature of trigger event, periodic or aperiodic
  - *rate:* periodic frequency if event is periodic, or average and/or maximal arrival rate if event is aperiodic

For the basic course and each of the alternate courses the following information is provided:

- course name: name for the course
- description: set of steps that take place during the course
- postcondition: conditions that exist after the steps outlined in the course are executed
- performance specifications (optional):
  - response
  - required response time
  - response type: hard or soft deadline
  - source of requirement: source of the performance requirement

The *Measurement-to-Track Association* use case is described in Figure 3 and the basic course—*Create New Track*—is described in Figure 4.

Use Case : 1 : Measurement-To-Track Association

Purpose

A measurement is received from a sensor through the communications link to be evaluated for potential association with an existing track.

Precondition

1. There is a set of tracks, AND
2. At a time t a measurement from a sensor is received through the communications link.

Requirements Satisfied by this Use Case

<None>

Basic Course :

Create New Track

Alternate Courses

Update Track
Filter Applied to Initial Tracks Returns No Tracks and Track is Created
Filter Applied to Propagated Tracks Returns No Tracks and Track is Created
Create Unassociated Measurement
Filter Applied to Initial Tracks Returns No Tracks and Track is Not Created
Filter Applied to Propagated Tracks returns No Tracks and Track is Not Created

Includes

-> 2: Filter Tracks
-> 3: Propagate Tracks

Included By

Extends

Extended By

Communicates With

-> 2: Comms Link

Figure 3: Measurement-to-Track Association Use Case

Use Case	: 1 : Measurement-To-Track Association
Course	: **Create New Track**

Description

1. Measurement collected by the sensor is reported to the system through the communications link.
2. Filter Tracks use case is invoked and a set of filtered tracks is returned.
3. Propagate Tracks use case is invoked so that the filtered tracks are propagated to measurement time t.
4. Filter Tracks use case is invoked again with the propagated tracks to make sure that the initial criteria still apply. Candidate tracks that do not match the filter criteria are eliminated.
5. Algorithms are applied to the candidate tracks to determine the likelihood that the measurement should be associated with a candidate track.
6. Because there were no candidate tracks that showed association values above the accepted threshold, an algorithm is applied to determine if the measurement provides additional data so that one can decide from looking at the set of yet unassociated measurements if there is enough information for a new track to be created.
7. Because there is enough information, a new track is created based on the reported measurement data plus the set of related unassociated measurements.
8. The measurement is added to the set of associated measurements and linked to the newly created track.

Postcondition

A new track is created and the measurement is added to the set of associated measurements and linked to the newly created track.

Figure 4: Create New Track—Basic Course for the Measurement-to-Track Association Use Case

The alternate courses for the *Measurement-to-Track Association* use case and the *Filter Tracks* and *Propagate Tracks* use cases are described in Appendix A on page 43.

3.3 Domains

Domains represent the different subject matter areas that we must understand to build a system [Mellor 02]. For the measurement-to-track association model problem there are two domains of interest: Track Management and Communications Interface.[4]

- Track Management: Provides the track storage as well as the track association, filtering, and propagation functionality.[5]
- Communications Interface: Receives and translates measurements received from sensors through the communications link.

Figure 5 shows the domain model for the measurement-to-track association model problem. The Track Management domain has a dependency on the Communications Interface domain. These domains most likely will include functionality to support other use cases. We are only interested in the functionality to support the measurement-to-track association use case. There are surely other domains of relevance to the whole system, but these are not included in the domain model because they are outside the scope of the model problem.

4. In Object-Oriented Analysis, domains are classified into four types according to the role each plays in the finished system: application domains, service domains, architectural domains, and implementation domains [Shlaer 92]. In this case, Track Management is an application domain and Communications Interface is a service domain.

5. It is also possible to separate the storage functionality from the association, filtering, and propagation functionality into different domains. For the purposes of this model problem, it does not make a difference and therefore we will consider them as part of the same domain.

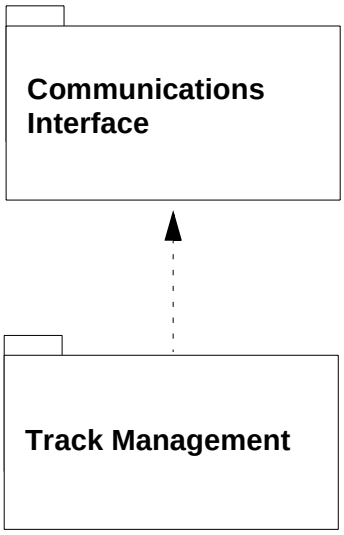

Figure 5: Domain Model

An alternative approach that considers locating measurement-to-track association in the *Sensor Management* domain is presented in Appendix D.

3.4 Sequence Diagrams for Domain Interaction Analysis

A *sequence diagram* presents the temporal interaction between objects as a set of exchanged messages. A sequence diagram has two dimensions. The vertical dimension represents time and the horizontal dimension represents the different objects that participate in the interaction. Normally time proceeds downward.

Another use for sequence diagrams is in Domain Interaction Analysis. *Domain Interaction Analysis* is a complementary technique that allows a project team to analyze the dynamics of the complete system of domains. These sequence diagrams are used to model the interactions between selected domains in order to satisfy the behavior specified in a particular use case [Kennedy 02].

The sequence diagrams at this level are extremely simple for the selected model problem, as can be seen in Figure 6. In this case the horizontal dimension represents domains instead of object instances. These would be more useful if we were modeling the whole system, where there is a greater level of interaction expected between domains. Section 3.6 presents sequence diagrams at a greater level of detail for object interaction analysis.

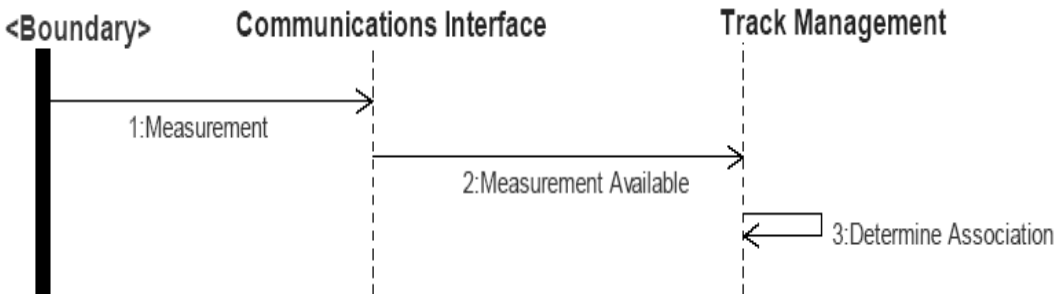

Figure 6: Sequence Diagram for the Measurement-to-Track Association Use Case

3.5 Class Diagram

A *class* represents a concept within the system being modeled. Classes contain data structure and behavior. A class diagram shows the static structure of the model, as a set of classes and relationships between classes and other elements of the model.

A number of classes were identified as part of the design process for the Track Management domain that handles the association of a measurement to a track. In the context of defining the model problem it is not important to correctly define all attributes for all classes.

A brief description of the classes in alphabetical order follows. It is important to note that even though the model problem only requires the application of filters to tracks, it is possible to apply filters to measurements as well. Because of this, the problem has been extended to observation management, where an observation refers either to a Track or a Measurement.[6]

- *Associated Measurement:* A measurement that has been associated to an existing track.
- *Candidate Observation*: When a filter is first applied either to a set of tracks or a set of measurements, a Candidate Observation is created for each track or measurement that matches the filter criteria. It is important to note that the *Candidate Observation* does not need to contain all the attributes of a measurement or track, only those that are necessary to make the association.
- *Filter*: Base class for all filters. At this point only one type of filter will be considered, but as will be seen in Section 4, other types of filters are possible.
- *Measurement*: All measurements provided by a sensor or communications interface and received by the *Observation Manager.* The attributes in this class have not been fully defined.

6. These two classes also have a large number of attributes in common.

- *Observation*: Object data managed within the Track Management domain—measurements and tracks.
- *Observation Manager*: Handles the sequence and setup of all the measurement-to-track association operations.
- *RBE Filter*: Given a relative position defined by range, bearing, elevation, and a radius, this type of filter will define a spherical volume. Application of the filter will then identify the tracks or measurements that lie within the volume determined by the filter.
- *Track*: State data about a track object. The attributes in this class have not been fully defined.
- *Track History*: History associated to a specific track.
- *Unassociated Measurement:* A measurement that has not yet been associated to an existing track.

The main class in the *Track Management* domain is *Track*. The *Track* class is a sub-class of the *Observation* class. This means that *Track* inherits all attributes and operations from *Observation*. A description of the Track class can be found in Table 1.

A description of the rest of the classes can be found in Appendix B on page 53.

Based on the classes identified previously, these can be combined in a class diagram. The class diagram in Figure 7 represents the classes and relationships necessary to implement the functionality required by the *Track Management* domain in reference to the *Measurement-to-Association* use case.[7]

The class diagram contains two basic constructs. These are

- Classes: These are denoted by the rectangular boxes in the diagram. Each box is divided into three parts. The top part contains the class name, the middle part contains the attributes that have been defined for the class, and the bottom part contains the operations that can be performed on instances of this class.
- Relations: The lines connecting the classes are relations. Relations are labeled by the modeling tool as R#. There are special types of relations, such as R2, that represent a superclass to sub-class relation (inheritance). Relations also show multiplicity and roles. Multiplicity represents the number of instances of each class that form part of the relation. Roles are the name of the part that each class plays in the relation.

As an example of how to interpret the diagram in Figure 7, the *Track* class is a subclass of the *Observation* class, as expressed by relation *R2*. The *R1* relation applies to the *Track* class

7. For diagram simplicity, all constructors and all operations that just “set” and “get” attribute values are omitted.

through the *Observation* class. The way to read R1 is "An *Observation Manager* manages one or many *Track*s; a *Track* is managed by one *Observation Manager*." Similarly, *R9* is read "A *Track* maintains zero or many *Track Histories*; a *Track History* is maintained by one *Track*.[8] Relation *R11* is read "A *Track* is associated to one or many *Associated Measurement*; an *Associated Measurement* is associated to one *Track*." During implementation, these *one or many* and *zero or many* relations can be translated into an associated list or some other data structure that represents this type of relationship between classes. For example, the *Track* class could have an additional attribute of type *List* that indicates the current relationship between *Track* and *Track History.* This *List* would contain elements of type *Track History.* Another way to represent this relation is as an attribute in the "one-or-many-side" class that refers to the "one-side" class. For example, *Associated Measurement* could have an attribute *TrackId* that represents the identification number of the associated track.

8. *"Track maintains zero or more Track History"* translates to "Each *Track* maintains zero or more instances of *Track History*."

*Table 1: Description of the **Track** Class*

Class Name	**Track**
Description	State data about a track object

Attributes

Name	**Description**	**Type**
id (from *Observation*)	Unique identification number for track	TrackID[a]
time (from *Observation*)	Time of latest observation associated with the track	Time of Day
latitude (from *Observation*)	Latitude of track	Real
longitude (from *Observation*)	Longitude of track	Real
altitude (from *Observation*)	Altitude of track	Real
velocity	Velocity for the track object	Real
type	Type of track; i.e. air, surface, ballistic, etc.	TrackType[b]

Operations

Name	Description	Parameters	Return Type
delete (from *Observation*)	Deletes a track	None	None
update	Updates a track with information from a given measurement	measurement: *Measurement*	None
create	Creates a track with information from a given measurement	measurement: *Measurement*	None
moveToHistory	Creates a copy of the track in *Track History* with its current values	None	None

a. TrackID is a user-defined type that needs to be specified.

b. TrackType is a user-defined type that needs to be specified.

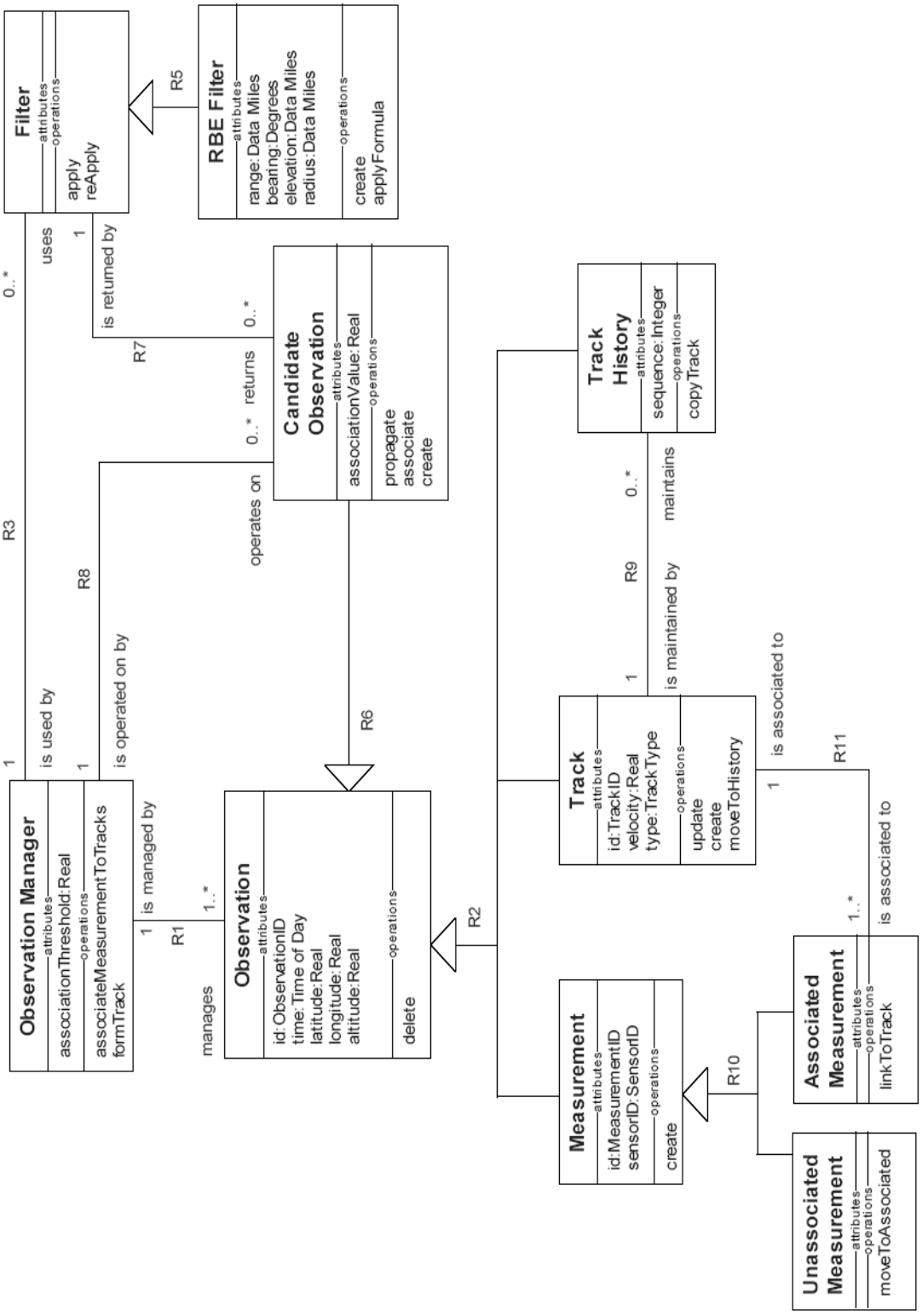

Figure 7: Class Diagram for the Track Management Domain

3.6 Sequence Diagrams for Object Interaction Analysis

As previously expressed, a sequence diagram presents the interaction between objects as a set of exchanged messages. A sequence diagram has two dimensions: the vertical dimension represents time, and the horizontal dimension represents the different object instances that participate in the interaction. Normally time proceeds downward.

The sequence diagrams in this section are at a different level of abstraction from those in Section 3.4.[9] The sequence diagram in Section 3.4 represents the interaction between domains to satisfy the behavior outlined by the *Measurement-to-Track Association* use case as well as a subset of the steps outlined in Section 2.2. The sequence diagrams in this section represent the interaction between objects in the Track Management domain to satisfy the different courses in the *Measurement-to-Track Association* use case. Figure 8 represents the basic course—*Create New Track*—and Figure 9 represents one of the alternate courses—*Update Track*.

Because all constructors and operations that just "set" and "get" values have been omitted from the class diagram, calls to these operations are represented as text over the arrow messages that represent the operation(s) that take place. For example, *Get Track Data* would translate into calls to all the necessary "get" operations.

The sequence diagrams for the rest of the alternate courses can be found in Appendix C on page 63.

9. The iUML tool selected by the customer does not use sequence diagrams within a domain. Instead, it uses one class collaboration diagram per domain, as presented in Section . We believe that sequence diagrams are useful for analysis at this level, which is why they were developed using Rational Rose and are included in this report.

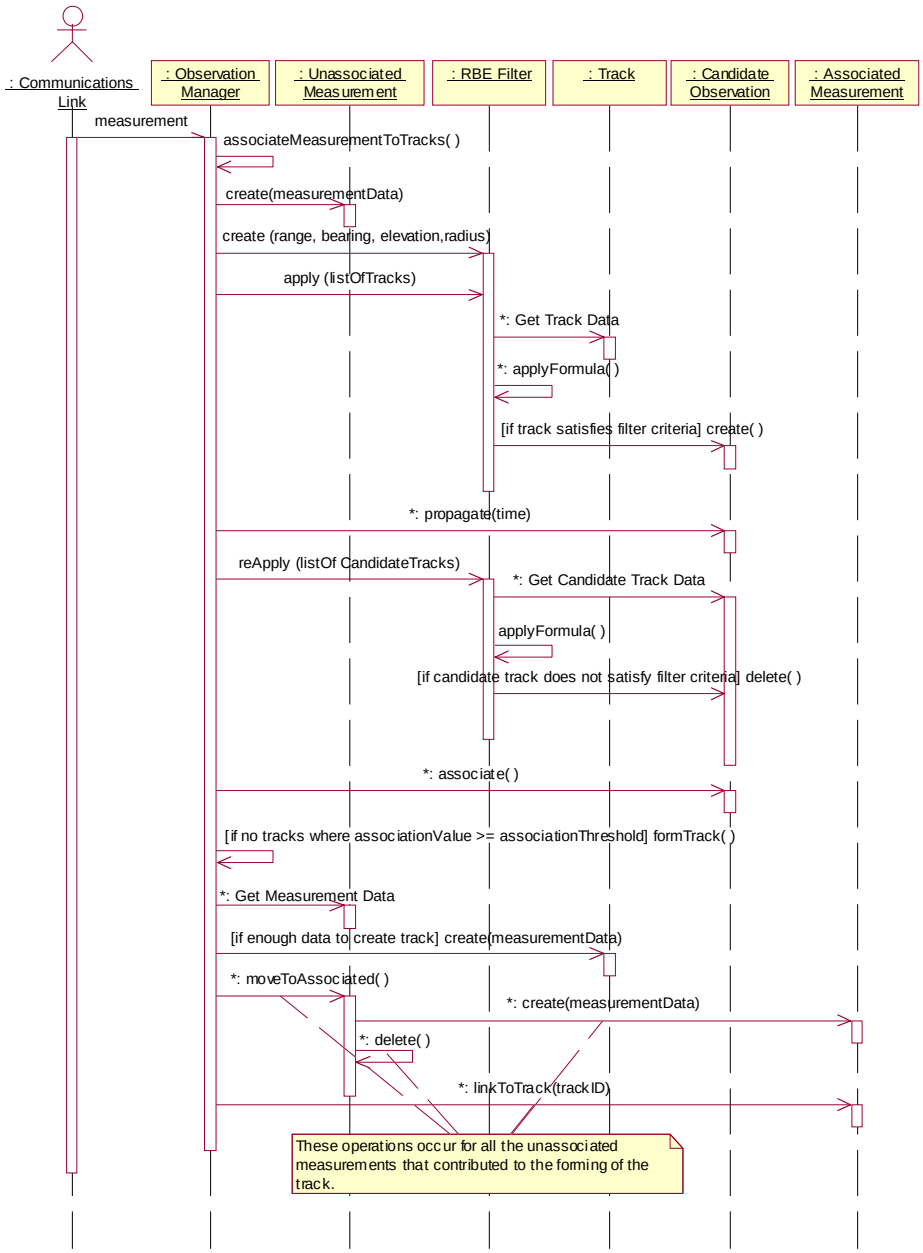

Figure 8: Sequence Diagram for the Measurement-to-Track Association Use Case Basic Course—Create New Track

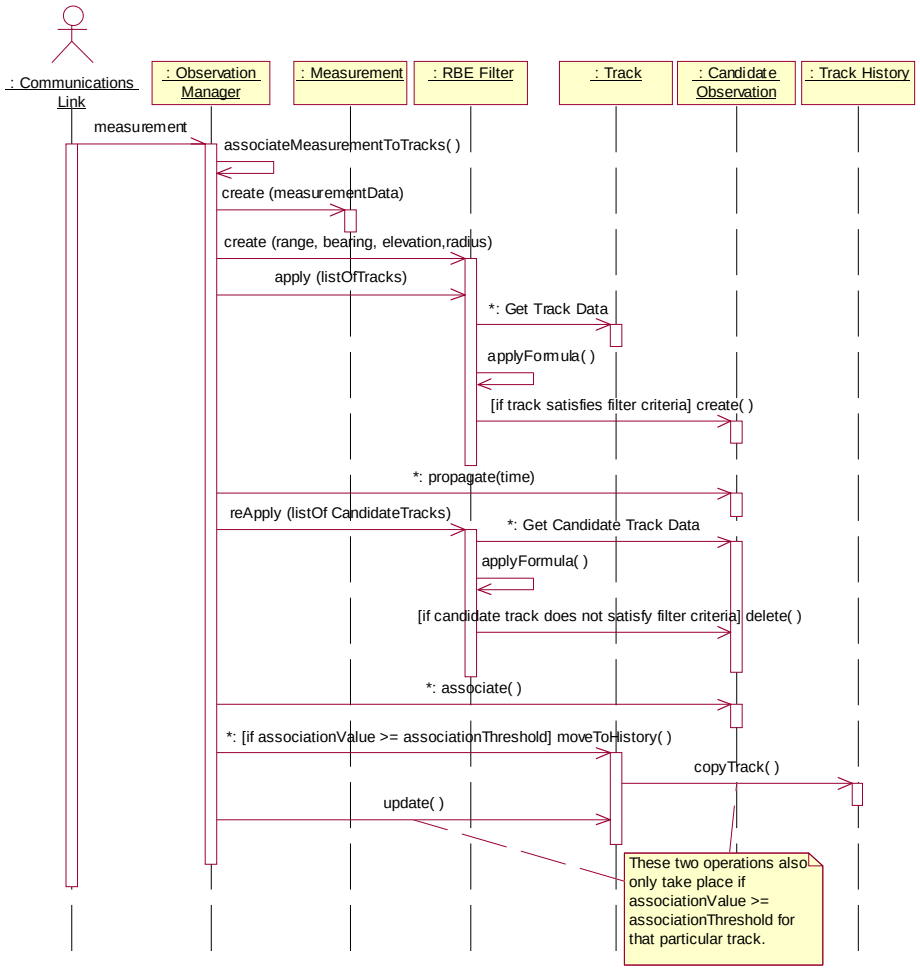

Figure 9: Sequence Diagram for the Measurement-to-Track Association Use Case Alternate Course—Update Track

3.7 Class Collaboration Diagram

A *class collaboration diagram* is a graphical summary of the interactions between the classes in a domain.[10] Messages with half arrowheads represent asynchronous signals from one state machine to another. Messages with full arrowheads represent synchronous operation invoc-

10. In traditional UML, a collaboration diagram represents the interaction between objects to satisfy a particular use case—similar to a sequence diagram but less tightly organized and with less emphasis on temporal sequence. Nonetheless, you can represent sequence and concurrency in a UML collaboration diagram. In the iUML tool a collaboration diagram is a more static view of what the responsibilities and interfaces of each class are and how classes will interact.

tions. *Terminators* are used to represent an abstraction of something outside the domain [Kennedy 02].

Figure 10 is the class collaboration diagram for the *Track Management* domain. *Comm Interface* is a terminator that represents another domain in the system that provides measurements asynchronously to the *Observation Manager* class.[11] All other classes communicate synchronously. For example, the diagram shows that the *Observation Manager* class can synchronously invoke the following operations on the *Track* class: *update, create, moveToHistory.*[12]

The material presented thus far offers an initial design of the model problem. It has been presented in the context of an object-oriented approach. We emphasize that the material presented here is an initial design. The determination of a final design is dependent upon many factors. In the present case we are interested in performance characteristics of the design. As noted earlier, performance properties will be viewed from two perspectives, namely qualitative and quantitative. The results of a performance model can then be used to assess the elements of the design described here. Such information will guide the final choice of a design for measurement-to-track association. It is important to note that this design and performance model for measurement-to-track association must be integrated into the context for a system solution.

11. The *Observation Manager* class is marked as a state machine because the iUML tool requires a class to be represented by a state machine in order to receive asynchronous messages.

12. Although a sub-class that inherits from a super-class inherits all its operations, the iUML tool does not allow it to show a class invoking inherited operations. For example, even though *Track* inherits the *delete* operation from *Observation*, it is not possible to show an invocation of the *delete* operation on *Track*.

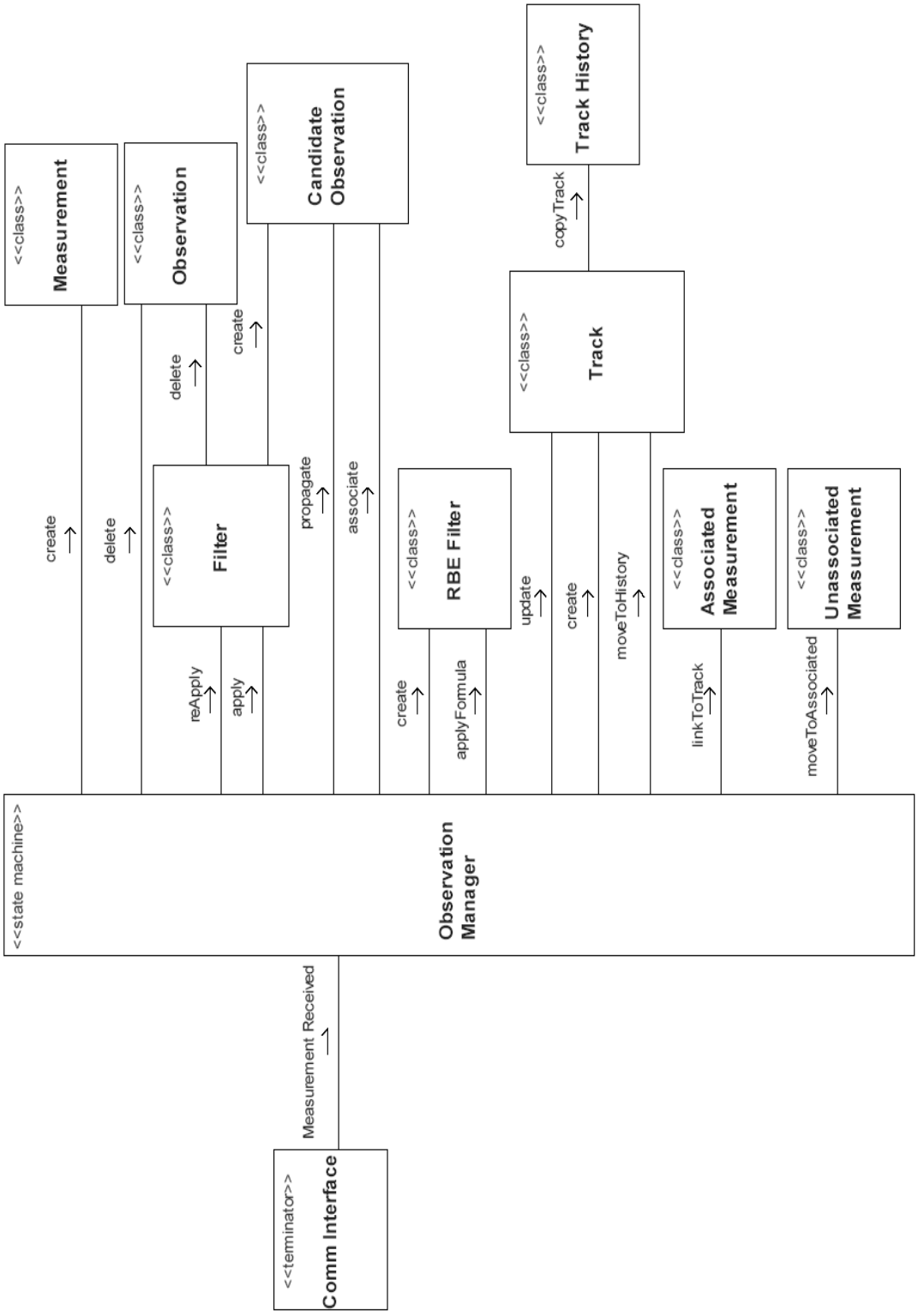

Figure 10: Class Collaboration Diagram for the Track Management Domain

4 Candidate Extensions

This section outlines several topics that can be further investigated as candidate extensions to the model problem.

4.1 Relation to Performance Considerations

We noted in Section 1 that this is the first in a series of reports dealing with a model problem. Our ultimate goals are two-fold. We want to develop a performance model of the design for the model problem of measurement-to-track association. The approach is outlined in Figure 11.

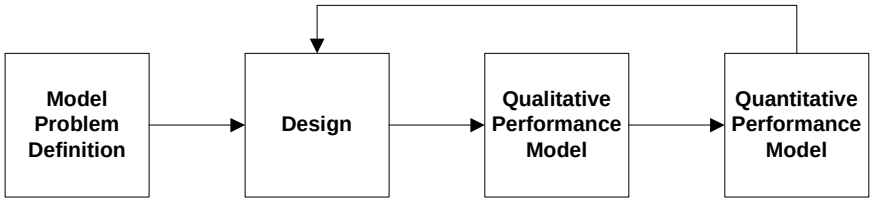

Figure 11: Overall Model Problem and Solution Context

For a given model problem, one may construct a design for its solution. An example of such a design is presented in Section 3 of this report. From a candidate design, one may construct a qualitative performance model. Such models are analytic in presentation and are intended to shed light on the performance behavior of the proposed solution. One can then elicit quantitative measurements and use them to assess the degree to which the design satisfies the problem at hand. A very important aspect of Figure 11 is the feedback from the quantitative performance model to the design process. In particular, it may be the case that quantitative results require some aspect of the design to be reconsidered. This implies that an iterative approach is essential to the development of a solution.

Note that the information contained in Figure 11 is presented in the context of a particular model problem. It does not show the fact that there can be many operations performed in the overall system. A systems-level performance model can be viewed as an integration of smaller performance models.

It is here that the choice of a model problem is of particular importance. Some general criteria for the selection of a model problem were discussed in Section 2.1. Of special importance is the degree to which the model problem may be reapplied in other contexts. For example, if there are many model problems that are similar in structure and function, though not necessarily in detail, to the model problem considered here, hopefully the same approach may be applied to other problems. Then, one would like to apply the performance models to these additional problems. Hence, a solution to a well chosen model problem can be applied to multiple instances of that model problem. We are, in effect, reusing a performance model by instantiating it in a different, though related context.

4.2 Other Filter Types

4.2.1 Geographic Filters

The RBE filter illustrated in Section 3.5 is an example of a geographic filter, but there are many other types of geographic filters. A simple example of another geographic filter is an LLE (latitude, longitude, elevation) filter, shown in Figure 12.

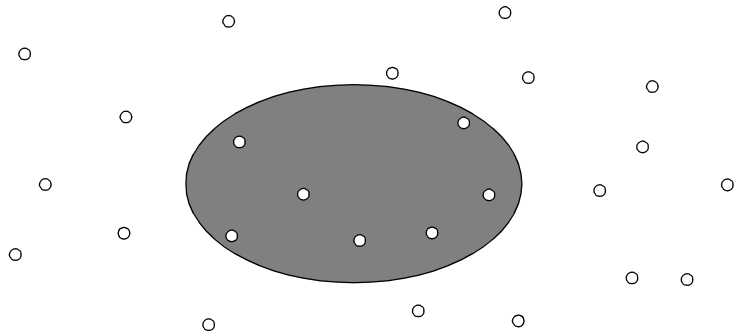

*Figure 12: Example of an **LLE Filter***

An LLE filter would determine the set of all objects of a particular type (such as tracks or measurement reports) that lie a certain distance from a specified absolute point. Notice that the LLE filter, described above, is similar, but not identical to the RBE filter, discussed earlier. In the case of the LLE filter, the volume of space defined by the filter is ellipsoidal, while the RBE filter defined a spherical volume of space. Another difference between the two types of filters is that the RBE filter defines a range about a relative point in space, while the LLE filter defines a range about an absolute point in space.

The representation shown in Figure 12 lends itself to the specification of a class of LLE filters. A description of such a class is shown in Figure 13.

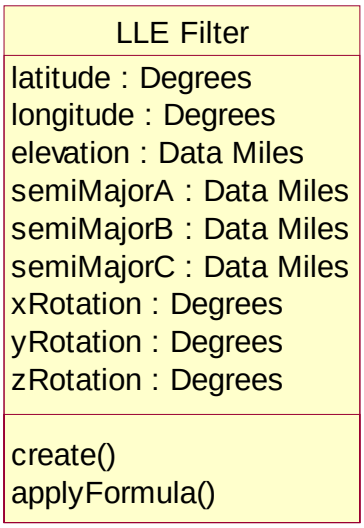

*Figure 13: Class Specification for **LLE Filter***

The LLE filter class has nine attributes that serve as filter criteria: *latitude*, *longitude*, *elevation*, *semiMajorA*, *semiMajorB*, *semiMajorC*, *xRotation*, *yRotation*, and *zRotation*.[13] Specification for the *Degrees* user-defined type is in Figure 14 and specification for the Data Miles user-defined type is in Figure 15. The *create* operation takes as parameters the values for the criteria and sets the respective attributes. The *applyFormula* operation applies the criteria to a given observation and returns a Boolean value indicating if the observation meets the criteria.

Type : Degrees

Base Type : Real

Constrained By
0.0 .. 360.0

Description
Represents degrees for angle values.

*Figure 14: Specification for the **Degrees** User-Defined Type*

13. We include the attributes x, y and z rotations as attributes to support an arbitrary position of the ellipsoid.

As another example of a geographic filter, consider the annular filter illustrated in Figure 16.

Type : Data Miles

Base Type : Real

Constrained By
>=0.0

Description
1 data mile(DM) = 6000 feet

*Figure 15: Specification for the **Data Miles** User-Defined Type*

The application of such a filter is to request all objects that are resident inside the annular area, as indicated by the shading in the filter. It may also possible to request all objects that are resident outside this annular area.

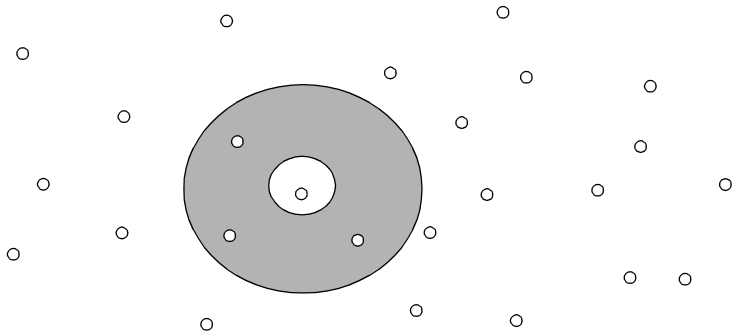

*Figure 16: Example of an **Annular Filter***

The representation shown in Figure 16 lends itself to the specification of a class of annular filters. A description of such a class is shown in Figure 17.

The annular filter class has six attributes that serve as filter criteria: *latitude, longitude, elevation, innerRadius, outerRadius,* and *constraint*. Specifications for the *Degrees* type, the *Data*

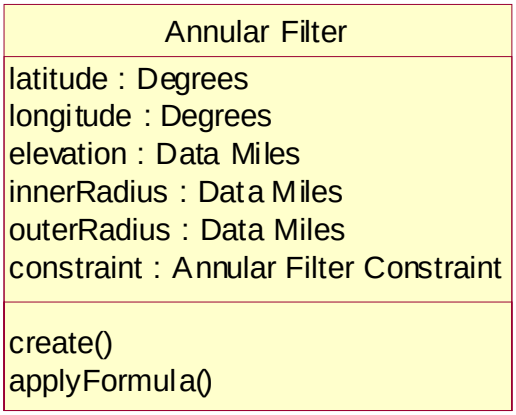

*Figure 17: Class Specification for **Annular Filter***

Miles type, and the *Annular Filter Constraint* type are in Figure 14, Figure 15, and Figure 18 respectively. The *create* operation takes as parameters the values for the criteria and sets the respective attributes. The *applyFormula* operation applies the criteria to the given observation and returns a Boolean value indicating if the observation meets the criteria.

Type : **Annular Filter Constraint**

Base Type : Enumeration

Constrained By
<Constraint>

Description
Indicates if the constraint for an annular filter is the region inside or outside of its boundaries.

Enumeration Values
INSIDE = 0
OUTSIDE = 1

*Figure 18: Specification for the **Annular Filter Constraint** User-Defined Type*

4.2.2 Generic Filters

The discussion of filters thus far has been focused on geographic filters. There are other types of filters that can be used; one type that we will introduce is a generic filter.

We define a generic filter to be one based on a vector defined by the triplet <attribute, operator, value>. The generic filter is relevant to a class specification, and may therefore be applied to an object, in the sense shown in Figure 19.

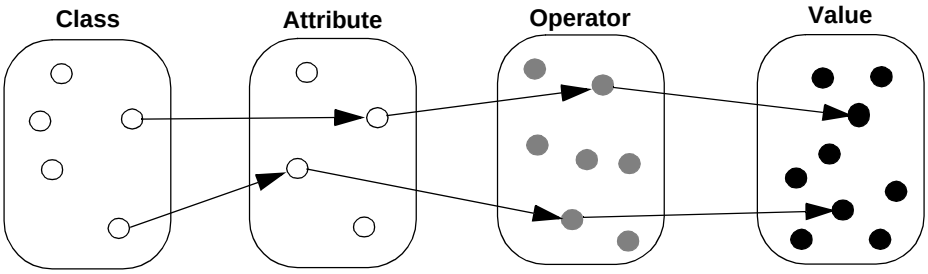

Figure 19: Relation of <Attribute, Operator, Value>

The choice of operators is dependent upon the data type of an attribute.[14] For example, for numeric data types, the usual operators of “equal,” “not equal,” “greater than,” and so on, apply. However, the values may also consist of character strings of various lineage (e.g., ASCII, ISO-10646, etc.). Furthermore, it is possible that an attribute could be a set, and that the operators are now those from set theory such as membership, or subset relation.

There is also the matter by which the elements of the tuples are connected. A simple approach might be to use a logical *AND* operator. Certainly other choices could be made and the choice of generality of permitted structure would no doubt depend on intended use.

To illustrate the application of a generic filter, suppose we wanted to determine all tracks such that their altitude is greater than 1000 miles. This is equivalent to the triple defined by

<attribute=altitude, operator=greater_than, value=1000)

The construction of a filter thus becomes one of identifying the elements of the tuple <attribute, operator, value>. The class specification for a generic filter appears in Figure 20.

14. An interesting occasion arises when we try to describe this problem in a formal manner. In particular, we have a set that contains values. It is clear that the set may contain values of different data types; for example, it may contain numeric types, or character types, or even sets. When describing the problem formally, we are faced with the need to describe a set whose members may be various data types. This is a departure from traditional set-theoretic approaches. Other work, not described here, is related to a formal specification approach that permits a set of mixed types. The approach is geared toward the description of dynamic systems and will be reported elsewhere.

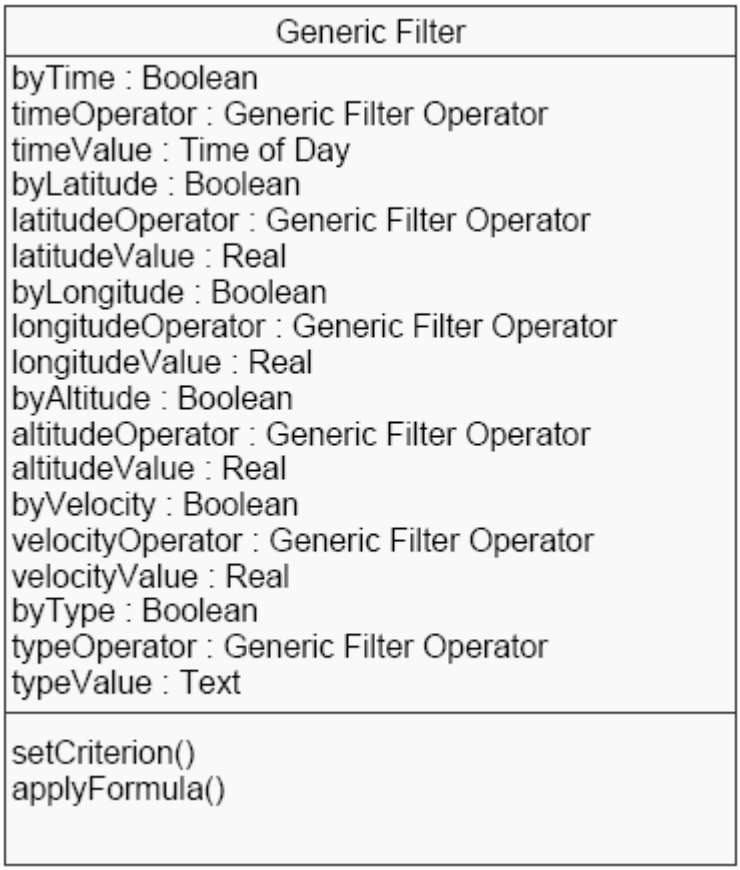

*Figure 20: Class Specification for **Generic Filter***

The specification for the user-defined type Generic Filter Operator is in Figure 21. This user-defined type determines the operator in the <attribute, operator, value> triplet.

The *setCriterion()* operation in this class takes as parameters an attribute/criterion of the user-defined type Filter Criterion, an operator of type Generic Filter Operator, and a value of the type of the criterion that is being set. The specification for the Filter Criterion user-defined type is in Figure 22.

For example, if a criterion for the filter is to identify those tracks whose altitude is greater than 1000 miles, the operation would be called *setCriterion(ALTITUDE, GREATER, 1000.0)*. The *setCriterion* operation would then set the attribute *byAltitude* to *True*, the attribute *altitudeOperator* to *GREATER*, and the attribute *altitudeValue* to *1000.0*.

Type : Generic Filter Operator

Base Type : Enumeration

Constrained By
<Constraint>

Description
Represents allowed operations for comparison of criteria within a generic filter.

Enumeration Values
EQUAL = 1
LESS = 2
GREATER = 3
LESS_OR_EQUAL = 4
GREATER_OR_EQUAL = 5
NOT_EQUAL = 6

*Figure 21: Specification for the **Generic Filter Operator** User-Defined Type*

Type : Filter Criterion

Base Type : Enumeration

Constrained By
<Constraint>

Description
Represents the attribute of the track to set as a criterion.

Enumeration Values
TIME = 1
LATITUDE = 2
LONGITUDE = 3
ALTITUDE = 4
VELOCITY = 5
TYPE = 6

*Figure 22: Specification for the **Filter Criterion** User-Defined Type*

This operation has to be overloaded so that there is a *setCriterion()* operation for each type of value (float, integer, string, etc.). Multiple criteria can be set by successive calls to the *setCriterion* operation. After all the criteria have been set, the *apply()* operation is invoked and applies the criteria to a set of observations.

The resulting class hierarchy for all filter types is shown in Figure 23.

4.2.3 Composition of Filters

A natural question to ask is the degree to which individual filters may be applied in succession. For example, Figure 24 shows the application of an annular filter, followed by the application of a rectangular filter. Each filter returns a set of values; in this case shown in Figure 24. We may represent the composition of filters as:

$$F = F_{ANNULAR} \cap F_{RECTANGULAR}$$

In other words the composition operator represents the intersection of the two filters. Other operators could be chosen; for example, instead of set-intersection, we could use set-union as the joining operator.

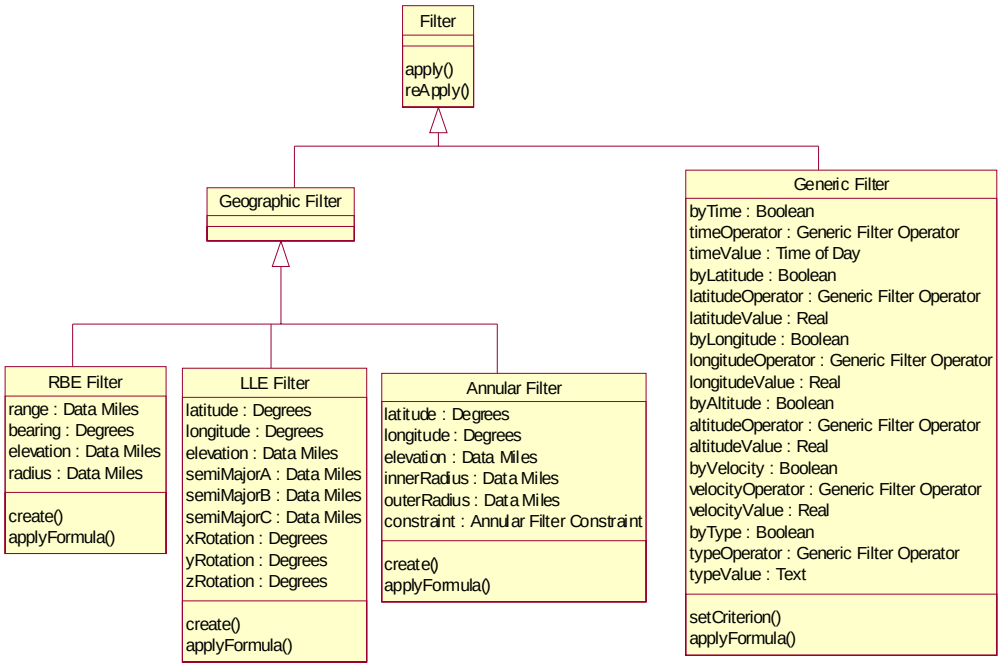

Figure 23: Class Hierarchy for Filters

The ability to apply multiple filters is recognized as having value. However, we do not believe that it represents any conceptual increase in the scope of the problem. Stated differently, the model problem of filtering can easily be extended to address the case of composition of filters.

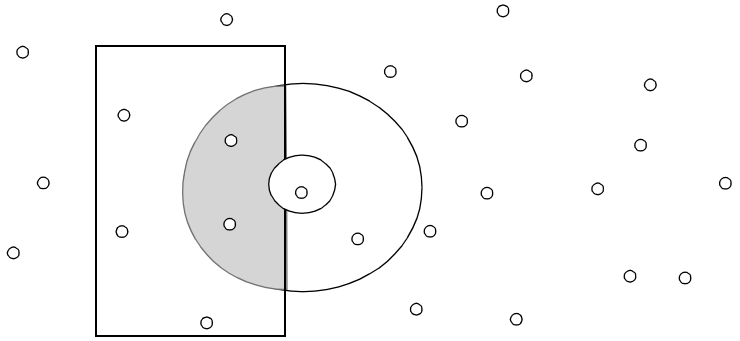

Figure 24: Composition of Annular and Rectangular Filter

4.2.4 Distribution of Filter Information

The scope for the work in this report is a distributed system of multiple platforms. Therefore, one might naturally question the ability to distribute filters among constituents of the larger system. For example, an RBE filter may be created in one context, and it is desirable to distribute that filter to some other context. We use context to refer to a process/processor combination.

We approach the distributed problem in terms of two dimensions. First, we are interested in the scope of distribution. In particular, we must consider system issues, and distribution across process and processor boundaries as defined by the context. The various contexts are shown in Figure 25.

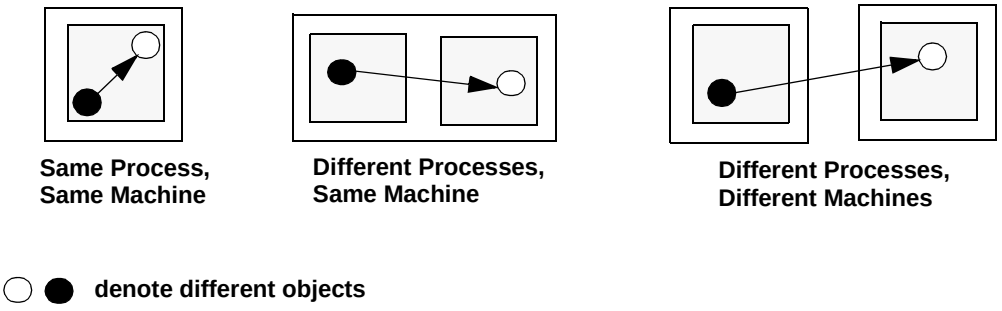

Figure 25: Contexts for Object Distribution

Second, we are interested in various mechanisms by which filters can be distributed. For distribution mechanisms we consider

- an object identifier, usually referred to as an object reference
- a message containing the relevant information about a filter
- a message containing a request for object creation

The resulting two-dimensional categorization is shown in Table 2.

Table 2: Category of Problems for Object Distribution

	Distribution Mechanism		
Scope of Distribution	Object Reference	Message	Object Creation
Same process, same machine			Not expected
Different processes, same machine			
Different processes, different machines			

As illustrated in Table 2, there are a variety of cases for object distribution. We will illustrate some of these cases, with a focus on the means of object distribution.

For distribution of objects in the same process on the same machine (such as between threads), a common approach is to use an identifier, or object reference, for the object of interest. Thus, one thread may pass an identifier to some object to a different thread, all within the same process on the same machine. The receiving thread may then invoke operations on the object.

However, when we move to the case where we wish to distribute some object across different machines, we are faced with potential challenges. For example, the underlying technology supporting the object system may not permit an object reference to be distributed (and subsequently accessed) across different processes, even in the same machine context. One way out of this difficulty is to use a message for the means of object distribution. The sequence of operations between a sender and a receiver of a filter, using a message-based approach might be the following:

- Sending System
  - invokes a method to create a message, based on the name of the class, and the values of the filter's attributes
  - initiates a communication with a receiving system
- Receiving System
  - unpacks and validates contents of the filter message

- instantiates an object of the specified class (in this case a filter) with the received information

Following the above sequence of operations, the receiving system may perform operations on the filter.[15] The operations are those specified in the class description. However, bringing in communication via messaging has opened up a number of questions such as quality of service, use of timeouts, and so on.

What happens if the receiving system does not have knowledge of the class? In this case, we must not only distribute attributes of a specified class, but all the relevant information about the class. We are, in effect, considering dynamic class creation. This is highly speculative and brings into question the use of mobile code and all its attendant issues.

The approach here has been viewed as a two-dimensional problem. In fact, it is possible to expand the scope to also include the underlying technology used to support objects and classes. In the simplest case, there may be interactions between two identical implementations, such as common object request broker architecture (CORBA). But there is also interest in a heterogeneous system where different implementations may be present. In such a case, one might be interested in interchange between a CORBA implementation and some other implementation.

Each of the preceding cases for object distribution constitutes a different level of approach to dealing with objects in a distributed system. Closely related to the design choices are those assumptions on the system partitioning that will be a constraint on any design approach. An example such as this is a potentially viable choice for a model problem in its own right!

4.3 Doctrine: An Example of Filter Application

We use the term *doctrine* to mean a set of rules that describe the behavior of some system. Such rules are important to the extent that they govern the system. In cases where doctrine is automated, without human intervention, it is especially important; one is turning over the behavior of a system to a machine.

Apart from machine execution of doctrine statements, such statements have utility from the perspective of changing the behavior of a system. For example, if an operator (either on the platform, or on some other platform) is able to change a doctrine statement, this can be represented as the change in the specification of the behavior. There is certainly value in the ability

15. Current systems achieve the equivalent behavior described here, although not in the context of an object-oriented approach. For example, given the attributes of a filter, they send a message to another system containing those attributes, and then the “filter” can be applied.

to dynamically change the behavior of some system without performing a software upgrade, for example.

As an illustration of a doctrine statement, with an eye toward the application of automated doctrine, consider the following:

```
If there is a TRACK
   range is less than 100 miles
   speed is greater than 600 miles/hour
   altitude is less than 100 feet
   type is unknown.
Then
   <perform_action>.
```

The preceding represents a doctrine statement simply as an if-then condition.[16] The text `<perform_action>` represents some action(s) that the system should take in the event that the if-clause is true. The details of the actions are not relevant to the remainder of this discussion and will be therefore omitted.

The if-clause represents a set of tests that are performed against a track. Note that the tests are based on attributes of a track (in this case, the track range, speed, altitude, and type). What is important to note is that the doctrine statement can be represented as a composition of two filters, namely

- a range filter, as discussed in connection with Figure 12 on page 28
- a generic filter based on track attributes speed, altitude, and type, as specified in Figure 20

We conclude that the if-clause of a doctrine statement can be considered as an application of a range filter and a generic filter. Recognizing this fact has two important consequences. First, it serves to illustrate the generality of the use of filters as part of a system design. Second, it further illustrates the generality of the model problem we have considered in this report. That is, to the extent that the model problem can be considered as the construction and application of a filter, so too does the problem apply to a subset of the processing performed for the application of doctrine.

4.4 Use of Object Query Language

The filters that have been described in this report have all been constructed in the context of an object-oriented development. That is, we have created classes for the filters, identified their

16. Notice that the overall time to process the query depends on the ordering of the search criteria. Query optimization is a common topic and concern for databases.

attributes, and illustrated ways that the filters can be applied. Filters are required to implement an operation called *applyFormula* that applies filter criteria to a given track.

It is worth noting that there is an alternative approach to applying the filter criteria to a set of tracks. We refer specifically to the Object Query Language (OQL) [ODMG 98]. OQL is a specification developed by the Object Data Management Group (ODMG) that allows one to perform queries on objects. It is an SQL-like declarative language with support for objects. It can be used in two different ways, either as an embedded function in a programming language or as an ad hoc query language. OQL works with programming languages for which ODMG has defined bindings, such as C++, Java, and SmallTalk. The advantage gained by using OQL is that it returns objects matching types in the specific programming language so that these objects can be easily manipulated.

Given the selected modeling tool, a question arises regarding how to include external libraries within the simulation environment and how to link generated code to external code. A mechanism is provided by the tool to interface to C programs. Further investigation would be required to determine if the tool can link to libraries in other languages different than C.

The topic of the use of OQL for track filtering, as opposed to direct creation and application of filter functions, is sufficiently broad to warrant consideration in its own right.

5 Summary

This report is the first in a series that illustrates the use of model problems to the design of a system. The selected model problem is measurement-to-track association and the development (and application) of filters as a selector mechanism. By using concepts and constructs from UML, Executable UML [Mellor 02], and Object-Oriented Analysis [Shlaer 92], the report presents an initial design of the model problem, as well as candidate extensions related to performance considerations, additional filter types, and the distribution of filter information. The next report in the series will explore performance properties of the initial design, namely qualitative and quantitative. The results of that performance model can then be used to assess the elements of the design described in this report.

Appendix A Additional Use Cases

This appendix contains the descriptions of the alternate courses for the *Measurement-to-Track Association* use case and the *Filter Tracks* and *Propagate Tracks* use cases.

Use Case : 1 : Measurement-To-Track Association
Course : Update Track

Description

1. Measurement collected by the sensor is reported to the system through the communications link.
2. Filter Tracks use case is invoked and a set of filtered tracks is returned.
3. Propagate Tracks use case is invoked so that the filtered tracks are propagated to measurement time t.
4. Filter Tracks use case is invoked again with the propagated tracks to make sure that the initial criteria still apply. Candidate tracks that do not match the filter criteria are eliminated.
5. Algorithms are applied to the candidate tracks to determine the likelihood that the measurement should be associated with a candidate track.
6. For the candidate track whose association value is larger than the acceptable threshold, corresponding track data is sent to track history and then updated with the reported measurement data.

Postcondition

For the candidate track whose association value is larger than the acceptable threshold, corresponding track data is sent to track history and then updated with the reported measurement data.

Figure 26: Update Track—Alternate Course for the Measurement-to-Track Association Use Case

Use Case : 1 : Measurement-To-Track Association
Course : Filter Applied to Initial Tracks Returns No Tracks and Track is Created

Description

1. Measurement collected by the sensor or communications interface is reported to the system through the communications link.
2. Filter Tracks use case is invoked and an empty set of filtered tracks is returned.
3. An algorithm is applied to determine if the measurement provides additional data so that one can decide from looking at the set of yet unassociated measurements if there is enough information for a new track to be created.
4. Because there is enough information, a new track is created based on the reported measurement data plus the set of related unassociated measurements.
5. The measurement is added to the set of associated measurements and linked to the newly created track.

Postcondition

A new track is created and the measurement is added to the set of associated measurements and linked to the newly created track.

Figure 27: Filter Applied to Initial Tracks Returns No Tracks and Track is Created—Alternate Course for the Measurement-to-Track Association Use Case

Use Case : 1 : Measurement-To-Track Association
Course : Filter Applied to Propagated Tracks Returns No Tracks and Track is Created

Description

1. Measurement collected by the sensor is reported to the system through the communications link.
2. Filter Tracks use case is invoked and a set of filtered tracks is returned.
3. Propagate Tracks use case is invoked so that the filtered tracks are propagated to measurement time t.
4. Filter Tracks use case is invoked again with the propagated tracks to make sure that the initial criteria still apply but no tracks remain after reapplying the filter.
5. An algorithm is applied to determine if the measurement provides additional data so that one can decide from looking at the set of yet unassociated measurements if there is enough information for a new track to be created.
6. Because there is enough information, a new track is created based on the reported measurement data plus the set of related unassociated measurements.
7. The measurement is added to the set of associated measurements and linked to the newly created track.

Postcondition

A new track is created and the measurement is added to the set of associated measurements and linked to the newly created track.

Figure 28: Filter Applied to Propagated Tracks Returns No Tracks and Track is Created—Alternate Course for the Measurement-to-Track Association Use Case

Use Case : 1 : Measurement-To-Track Association
Course : Create Unassociated Measurement

Description

1. Measurement collected by the sensor is reported to the system through the communications link.
2. Filter Tracks use case is invoked and a set of filtered tracks is returned.
3. Propagate Tracks use case is invoked so that the filtered tracks are propagated to measurement time t.
4. Filter Tracks use case is invoked again with the propagated tracks to make sure that the initial criteria still apply. Candidate tracks that do not match the filter criteria are eliminated.
5. Algorithms are applied to the candidate tracks to determine the likelihood that the measurement should be associated with a candidate track.
6. Because there were no candidate tracks that showed association values above the accepted threshold, an algorithm is applied to determine if the measurement provides additional data so that one can decide from looking at the set of yet unassociated measurements if there is enough information for a new track to be created.
7. Because there is not enough information to create a new track the measurement is added to the list of unassociated measurements.

Postcondition

The measurement is added to the list of unassociated measurements.

Figure 29: Create Unassociated Measurement—Alternate Course for the Measurement-to-Track Association Use Case

Use Case : 1 : Measurement-To-Track Association
Course : Filter Applied to Initial Tracks Returns No Tracks and Track is Not Created

Description

1. Measurement collected by the sensor or communications interface is reported to the system through the communications link.
2. Filter Tracks use case is invoked and an empty set of filtered tracks is returned.
3. An algorithm is applied to determine if the measurement provides additional data so that one can decide from looking at the set of yet unassociated measurements if there is enough information for a new track to be created.
4. Because there is not enough information to create a new track the measurement is added to the list of unassociated measurements.

Postcondition

The measurement is added to the list of unassociated measurements.

Figure 30: Filter Applied to Initial Tracks Returns No Tracks and Track is Not Created—Alternate Course for the Measurement-to-Track Association Use Case

Use Case : 1 : Measurement-To-Track Association
Course : Filter Applied to Propagated Tracks Returns Nc Tracks and Track is Not Created

Description

1. Measurement collected by the sensor is reported to the system through the communications link.
2. Filter Tracks use case is invoked and a set of filtered tracks is returned.
3. Propagate Tracks use case is invoked so that the filtered tracks are propagated to measurement time t.
4. Filter Tracks use case is invoked again with the propagated tracks to make sure that the initial criteria still apply but no tracks remain after reapplying the filter.
5. An algorithm is applied to determine if the measurement provides additional data so that one can decide from looking at the set of yet unassociated measurements if there is enough information for a new track to be created.
6. Because there is not enough information to create a new track the measurement is added to the list of unassociated measurements.

Postcondition

The measurement is added to the list of unassociated measurements.

Figure 31: Filter Applied to Propagated Tracks Returns No Tracks and Track is Not Created—Alternate Course for the Measurement-to-Track Association Use Case

Use Case : 2 : Filter Tracks

Purpose
A set of tracks is examined to determine if they match the set of criteria specified by filter.

Precondition
There is a set of tracks.

Requirements Satisfied by this Use Case
<None>

Basic Course :
Filter Tracks and Create Copies of Matching Tracks

Alternate Courses
Refilter Tracks and Remove Non-Matching Tracks

Includes

Included By
-> 1: Measurement-To-Track Association

Extends

Extended By

Communicates With

Figure 32: Filter Tracks Use Case

Use Case : 2 : Filter Tracks
Course : Filter Tracks and Create Copies of Matching Tracks

Description

1. A filter is created with values for its criteria.
2. The filter is given a set of tracks of tracks.
3. An empty list of candidate tracks is created.
4. For each track in the set of tracks
   4.1. Track attribute values are compared to filter criteria
   4.2. If the track satisfies the filter criteria it is copied and added to the list of candidate tracks.
5. Set of candidate tracks is returned.

Postcondition

Set of candidate tracks is returned.

Figure 33: Filter Tracks and Create Copies of Matching Tracks—Basic Course for the Filter Tracks Use Case

Use Case : 2 : Filter Tracks
Course : Refilter Tracks and Remove Non-Matching Tracks

Description

1. An existing filter is given a set of tracks of tracks.
2. For each track in the set of tracks
   2.1. Track attribute values are compared to filter criteria.
   2.2. If the track does not satisfy the filter criteria it is removed from the list of tracks.
3. Set of refiltered candidate tracks is returned.

Postcondition

Set of refiltered candidate tracks is returned.

Figure 34: Refilter Tracks and Remove Non-Matching Tracks—Alternate Course for the Filter Tracks Use Case

Use Case : 3 : Propagate Tracks

Purpose
Tracks are propagated to a given time t.

Precondition
There is a set of tracks.

Requirements Satisfied by this Use Case
<None>

Basic Course :
Propagate Tracks

Alternate Courses

Includes

Included By
-> 1: Measurement-To-Track Association

Extends

Extended By

Communicates With

Figure 35: Propagate Tracks Use Case

Use Case : 3 : Propagate Tracks
Course : Propagate Tracks

Description
For each set in a set of tracks:
1. Apply propagation algorithm
2. Update track attribute values correspondingly

Postcondition
Tracks in the set are updated to reflect propagation to time t.

Figure 36: Propagate Tracks—Basic Course for the Propagate Tracks Use Case

Appendix B Details of Classes

This appendix contains the description of the classes in the Track Management domain in alphabetical order.

*Table 3: Description of the **Associated Measurement** Class*

Class Name	**Associated Measurement**		
Description	A measurement that has been associated to an existing track		
Attributes			
Name	*Description*	*Type*	
id (from *Observation*)	Unique identification number for measurement	MeasurementID[a]	
time (from *Observation*)	Time measurement was made	Time of Day	
latitude (from *Observation*)	Latitude of measurement	Real	
longitude (from *Observation*)	Longitude of measurement	Real	
altitude (from *Observation*)	Altitude of measurement	Real	
sensorId (from *Measurement*)	Identification of the sensor that received the measurement	SensorID[b]	
Operations			
Name	*Description*	*Parameters*	*Return Type*
delete (from *Observation*)	Deletes a measurement	None	None
create (from *Measurement*)	Creates a measurement with the received measurement data	One parameter per measurement attribute	None
linkToTrack	Links the associated measurement to the given track	track:*Track*	None

a. MeasurementID is a user-defined type that needs to be specified.

b. SensorID is a user-defined type that needs to be specified.

*Table 4: Description of the **Candidate Observation** Class*

Class Name	**Candidate Observation**		
Description	When a filter is first applied either to a set of tracks or a set of measurements, a Candidate Observation is created for each track or measurement that matches the filter criteria.		
Attributes			
Name	*Description*	*Type*	
id (from *Observation*)	Unique identification number for the candidate observation	ObservationID[a]	
time (from *Observation*)	In case of measurement, time observation is received; in case of track, time of latest observation associated with the track	Time of Day	
latitude (from *Observation*)	Latitude of observation	Real	
longitude (from *Observation*)	Longitude of observation	Real	
altitude (from *Observation*)	Altitude of observation	Real	
associationValue	Association value between an observation and a given measurement	Real	
Operations			
Name	*Description*	*Parameters*	*Return Type*
delete (from *Observation*)	Deletes a candidate observation	None	None
propagate	Propagates a candidate observation to time *t* and updates attribute values accordingly	t: *Time Of Day*	None
associate	Associates a candidate observation to a given measurement and updates the associationValue attribute	measurement: *Measurement*	None
create	Creates a candidate observation by creating a copy of the given observation	observation: Observation	None

a. ObservationID is a user-defined type that needs to be specified.

*Table 5: Description of the **Filter** Class*

Class Name	**Filter**		
Description	Base class for all filters. A filter applies a set of criteria to a set of observations to determine if they meet the filter criteria.		
Attributes			
Name	*Description*	*Type*	
Operations			
Name	*Description*	*Parameters*	*Return Type*
apply	Applies a filter to a set of observations and creates a set of candidate observations by copying those that match the criteria	listOfObservations: *Observation*	listOfCandidateObservations: *Candidate Observation*
reapply	Applies the filter to a set of candidate observations and deletes those that do not satisfy the filter criteria	listOfCandidateObservations: *Candidate Observation*	newListOfCandidateObservations: *Candidate Observation*

*Table 6: Description of the **Measurement** Class*

Class Name	**Measurement**		
Description	All measurements provided by a sensor or communications interface and received by the Track Management domain		
Attributes			
Name	*Description*	*Type*	
id (from *Observation*)	Unique identification number for measurement	MeasurementID[a]	
time (from *Observation*)	Time measurement was made	Time of Day	
latitude (from *Observation*)	Latitude of measurement	Real	
longitude (from *Observation*)	Longitude of measurement	Real	
altitude (from *Observation*)	Altitude of measurement	Real	
sensorId	Identification of the sensor that received the measurement	SensorID[b]	
Operations			
Name	*Description*	*Parameters*	*Return Type*
delete (from *Observation*)	Deletes a measurement	None	*None*
create	Creates a measurement with the received measurement data	One parameter per measurement attribute	*None*

a. MeasurementID is a user-defined type that needs to be specified.

b. SensorID is a user-defined type that needs to be specified.

*Table 7: Description of the **Observation** Class*

Class Name	**Observation**		
Description	Object data managed within the Track Management domain—measurements and tracks		
Attributes			
Name	*Description*	*Type*	
id	Unique identification number for the observation[a]	ObservationID	
time	In case of measurement, time observation is made; in case of track, time of latest observation associated with the track	Time of Day	
latitude	Latitude of observation	Real	
longitude	Longitude of observation	Real	
altitude	Altitude of observation	Real	
Operations			
Name	*Description*	*Parameters*	*Return Type*
delete	Deletes an observation	*None*	*None*

a. This attribute is overridden by the id attribute in its subclasses: Measurement, Track, Track History.

*Table 8: Description of the **Observation Manager** Class*

Class Name	**Observation Manager**		
Description	Handles the sequence and setup of all the measurement-to-track association operations		
Attributes			
Name	*Description*	*Type*	
associationThreshold	Threshold association value for determining whether a measurement is associated to a track	Real	
Operations			
Name	*Description*	*Parameters*	*Return Type*
associateMeasurement ToTracks	Handles the sequence and setup of the operations that take place is measurement-to-track association	measurementData: Undefined[a]	None
formTrack	Determines if there is enough information in the set of unassociated measurements that can be combined with the received measurement to form a track	None	None

a. The type of parameter *measurementData* will depend on how the signal containing the measurement data is structured.

*Table 9: Description of the **RBE Filter** Class*

Class Name	**RBE Filter**		
Description	Given a position defined by range, bearing, and elevation, this type of filter will define a volume about that position.		
Attributes			
Name	*Description*	*Type*	
range	Range criteria for filter	Data Miles[a]	
bearing	Bearing criteria for filter	Degrees[b]	
elevation	Elevation criteria for filter	Data Miles	
radius	Radius criteria for filter	Data Miles	
Operations			
Name	*Description*	*Parameters*	*Return Type*
apply (from *Filter*)	Applies the RBE filter to a set of observations and creates a set of candidate observations by copying those that match the criteria	listOfObservations: *Observation*	listOfCandidateObservations: Candidate Observation
reapply (from *Filter*)	Applies the filter to a set of candidate observations and deletes those that do not satisfy the filter criteria	listOfCandidateObservations: *Candidate Observation*	newListOfCandidateObservations: Candidate Observation
create	Creates an RBE filter by setting the range, bearing, elevation, and radius attributes with the given values	range: Data Miles, bearing: Degrees, elevation: Data Miles, radius: Data Miles	None
applyFormula	Applies the filter specific formula to a given observation	observation: *Observation*	meetsCriteria: *Boolean*

a. The Data Miles user-defined type is specified in Section 4.2, Figure 15.

b. The Degrees user-defined type is specified in Section 4.2, Figure 14.

*Table 10: Description of the **Track** Class*

Class Name	**Track**		
Description	State data about a track object		
Attributes			
Name	*Description*	*Type*	
id (from *Observation*)	Unique identification number for track	TrackID[a]	
time (from *Observation*)	Time of latest observation associated with the track	Time of Day	
latitude (from *Observation*)	Latitude of track	Real	
longitude (from *Observation*)	Longitude of track	Real	
altitude (from *Observation*)	Altitude of track	Real	
velocity	Calculated velocity for the track object	Real	
type	Type of track; i.e. air, surface, ballistic, etc.	TrackType[b]	
Operations			
Name	*Description*	*Parameters*	*Return Type*
delete (from *Observation*)	Deletes a track	None	None
update	Updates a track with information from a given measurement	measurement: *Measurement*	None
create	Creates a track with information from a given measurement	measurement: *Measurement*	None
moveToHistory	Creates a copy of the track in *Track History* with its current values	*None*	*None*

a. TrackID is a user-defined type that needs to be specified.

b. TrackType is a user-defined type that needs to be specified.

*Table 11: Description of the **Track History** Class*

Class Name	**Track History**		
Description	State data about an object		
Attributes			
Name	*Description*	*Type*	
id (from *Track*)	Identification number of associated track	TrackID	
time (from *Observation*)	Time of observation associated with the track	Time of Day	
latitude (from *Observation*)	Latitude of track	Real	
longitude (from *Observation*)	Longitude of track	Real	
altitude (from *Observation*)	Altitude of track	Real	
sequence	Marks the sequence of the track in its history	Real	
Operations			
Name	*Description*	*Parameters*	*Return Type*
delete (from *Observation*)	Deletes a track from its history	None	None
copyTrack	Copies the given track information to the track history and assigns a sequence number	track: Track	None

*Table 12: Description of the **Unassociated Measurement** Class*

Class Name	**Unassociated Measurement**		
Description	A measurement that has not yet been associated to an existing track		
Attributes			
Name	*Description*	*Type*	
id (from *Observation*)	Unique identification number for measurement	MeasurementID[a]	
time (from *Observation*)	Time measurement was made	Time of Day	
latitude (from *Observation*)	Latitude of measurement	Real	
longitude (from *Observation*)	Longitude of measurement	Real	
altitude (from *Observation*)	Altitude of measurement	Real	
sensorId (from *Measurement*)	Identification of the sensor that received the measurement	SensorID[b]	
Operations			
Name	*Description*	*Parameters*	*Return Type*
delete (from *Observation*)	Deletes a measurement	None	None
create (from *Measurement*)	Creates a measurement with the received measurement data	One parameter per measurement attribute	None
moveToAssociated	Converts the unassociated measurement to an associated measurement	None	None

a. MeasurementID is a user-defined type that must be specified.

b. SensorID is a user-defined type that must be specified.

Appendix C Additional Sequence Diagrams

This appendix contains the sequence diagrams for the following *Measurement-To-Track Association* use case alternate courses:

- Filter Applied to Initial Tracks Returns No Tracks and Track is Created
- Filter Applied to Propagated Tracks Returns No Tracks and Track is Created
- Create Unassociated Measurement
- Filter Applied to Initial Tracks Returns No Tracks and Track is Not Created
- Filter Applied to Propagated Tracks Returns No Tracks and Track is Not Created

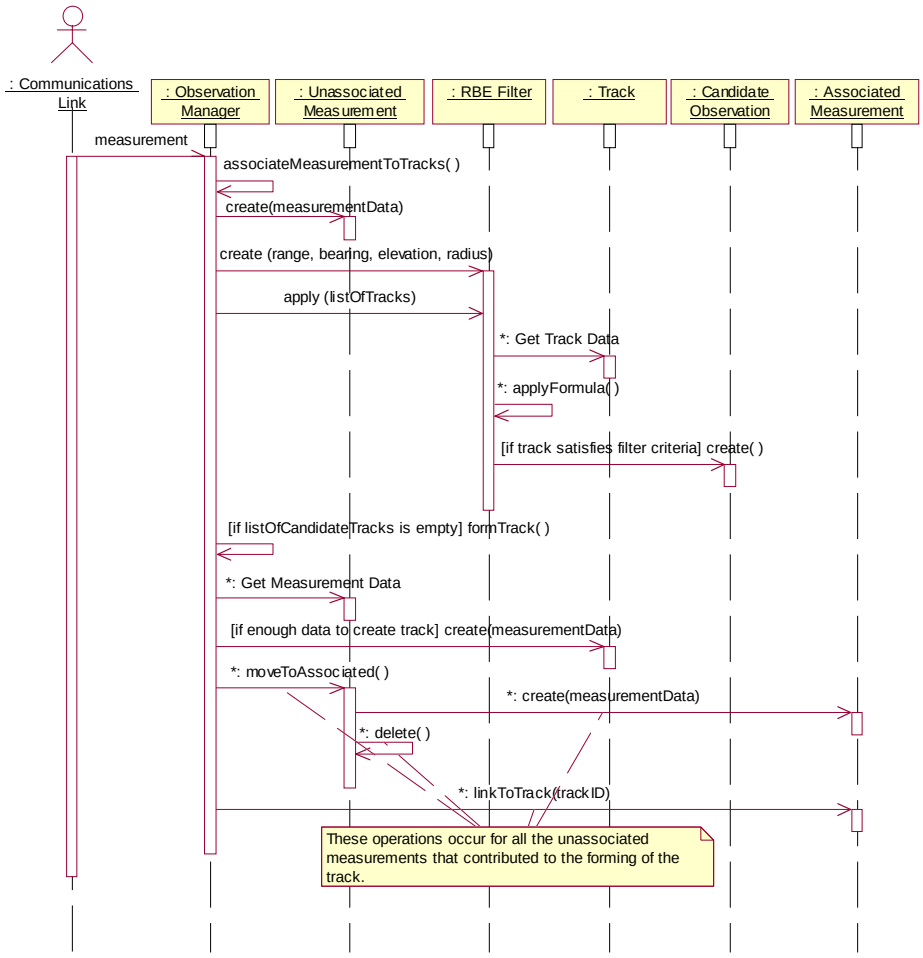

Figure 37: Sequence Diagram for the Measurement-to-Track Association Use Case Alternate Course—Filter Applied to Initial Tracks Returns No Tracks and Track Is Created

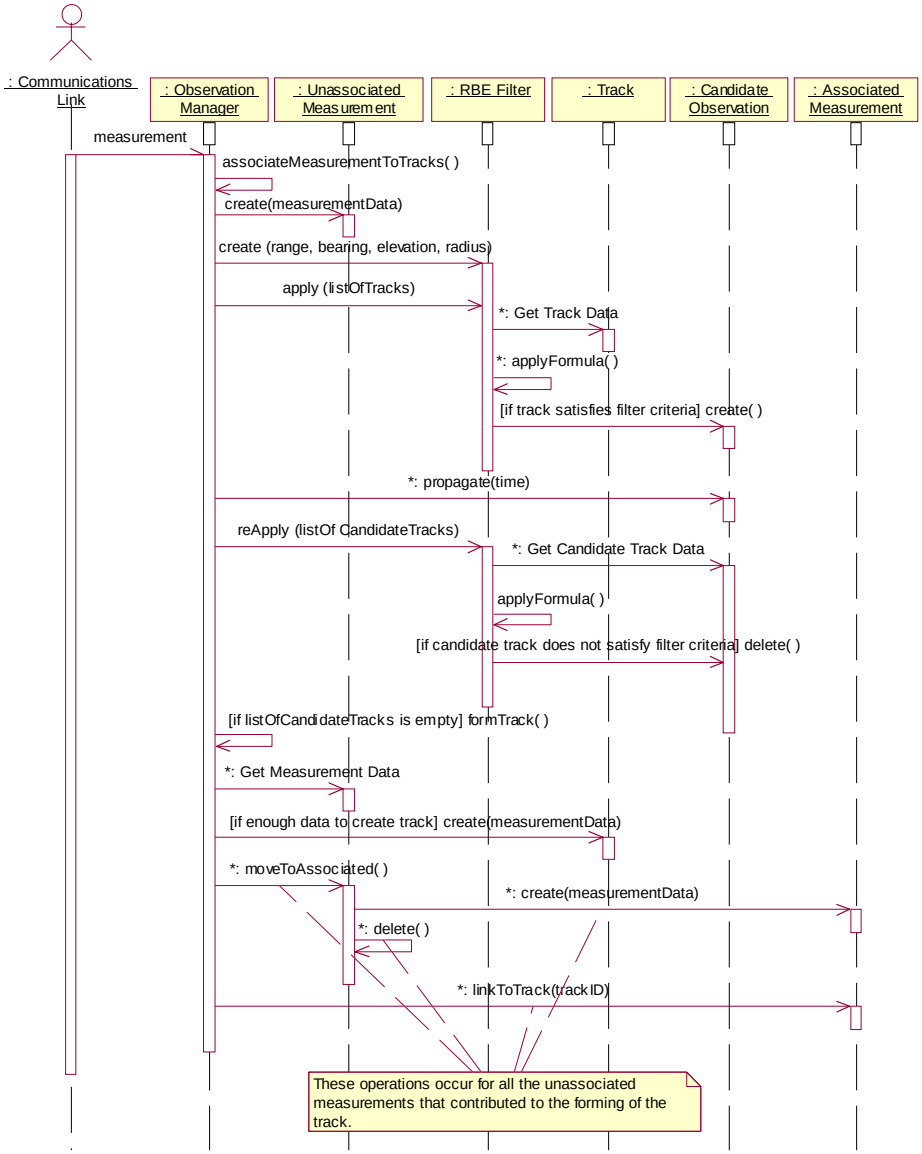

Figure 38: Sequence Diagram for the Measurement-to-Track Association Use Case Alternate Course—Filter Applied to Propagated Tracks Returns No Tracks and Track Is Created

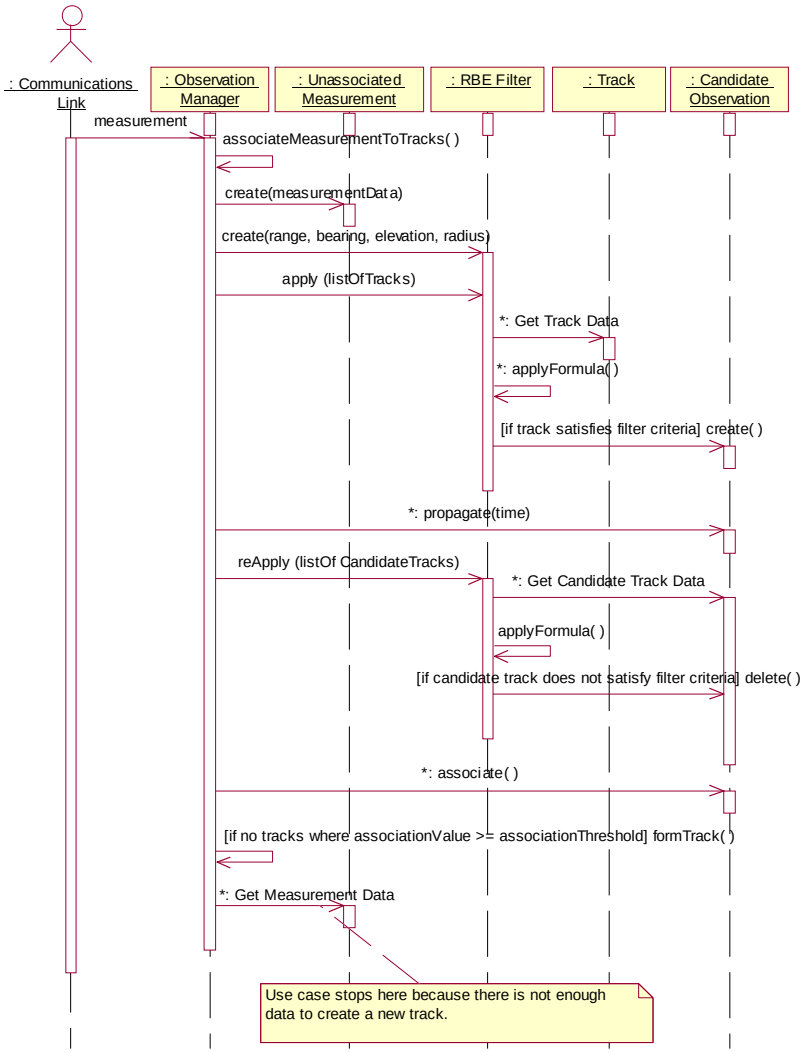

Figure 39: Sequence Diagram for the Measurement-to-Track Association Use Case Alternate Course—Create Unassociated Measurement

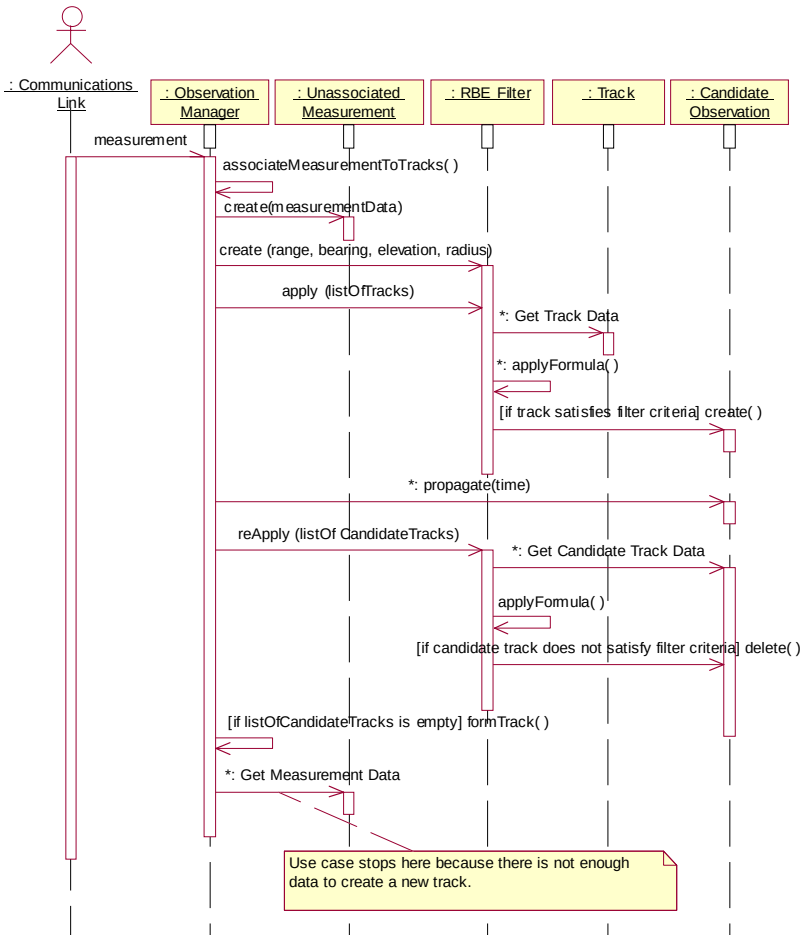

Figure 40: Sequence Diagram for the Measurement-to-Track Association Use Case Alternate Course—Filter Applied to Initial Tracks Returns No Tracks and Track Is Not Created

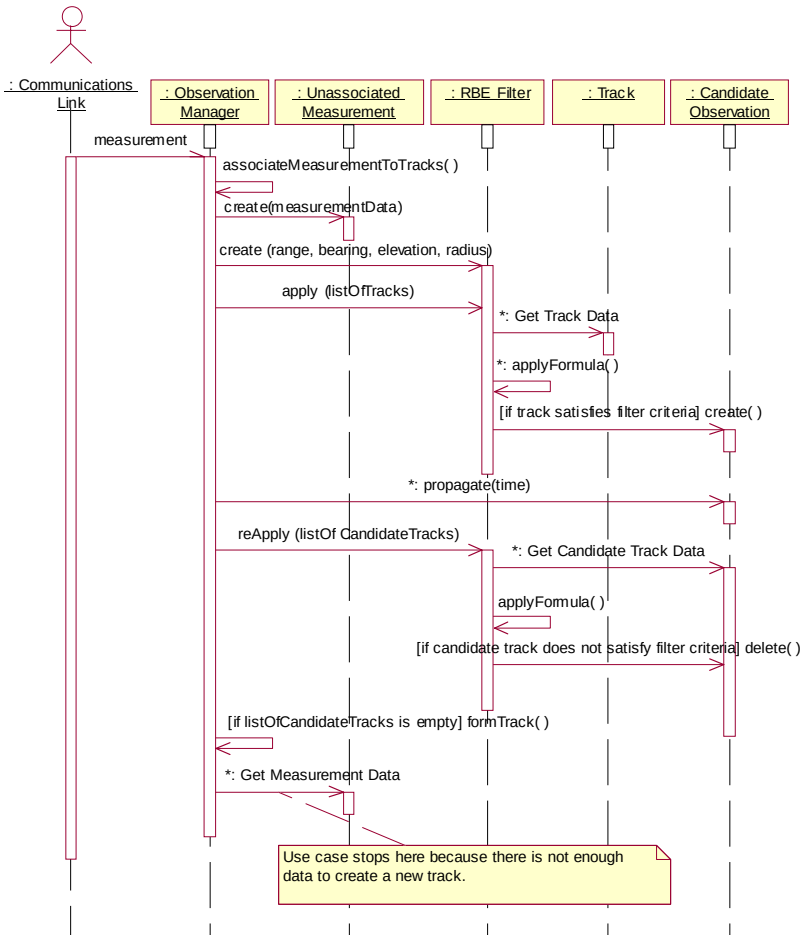

Figure 41: Sequence Diagram for the Measurement-to-Track Association Use Case Alternate Course—Filter Applied to Propagated Tracks Returns No Tracks and Track Is Not Created

Appendix D Measurement-to-Track Association in the Sensor Management Domain

The measurement-to-track association functionality in the model problem has been assigned to the Track Management domain because of the tight coupling between this function and the track and measurement data. There has been some discussion as to whether the measurement-to-track association should be assigned to the Sensor Management domain. The rationale for this alternative is that the algorithms for association are expected to be dependent on the type of sensor and therefore it would be better to maintain these algorithm differences in the Sensor Management domain. The problem with this approach is the access to track data that would still reside in the Track Management domain.

This appendix discusses the potential changes to the design of the model problem as well as the consequences of placing measurement-to-track association in the Sensor Management domain.

D.1 Changes to the Design of the Model Problem

The changes to the design of the model problem are minimal since domains are essentially sets of closely related objects that are treated as a unit for purposes of analysis. In other words, they correspond to the way that the project has decided to divide and allocate functionality in a system. The changes to the design of the model problem would be

1. Move the classes that contain the measurement-to-track association functionality in the Track Management domain to the Sensor Management domain.
2. Define the relationships between the newly moved classes and the classes that already exist in the Sensor Management domain.
3. Decide on the mechanism for accessing track data and make the appropriate changes to the model.

This last point is critical and will be the focus of the next section.

D.2 Consequences of Measurement-to-Track Association in the Sensor Management Domain

The separation between the measurement-to-track association functionality and the track data introduces a design problem for which there are two alternatives:

1. Keep the track data in the Track Management domain and have the Sensor Management domain communicate with the Track Management domain every time it needs access to data.
2. Maintain copies of the track data in the Sensor Management domain.

D.2.1 Track Data in Track Management

This option has the advantage of not having to maintain replicated track data, but has a potential for performance problems if the Track Management domain and the Sensor Management domain reside on different machines. These potential performance problems are caused by the tight coupling between track data and measurement-to-track association, as can be seen in the sequence diagrams in Section 3.6 and Appendix C.

If this is the option selected, the domain model would have to be updated to include the communication between the Track Management and Sensor Management domains, and the sequence diagrams and class collaboration diagram would also have to be updated to show this communication.

D.2.2 Track Data Replicated in Sensor Management

This option has the advantage of keeping track data local to the measurement-to-track association functionality but has challenges introduced by replication, such as data consistency. If a copy of the track data is going to be maintained in Sensor Management, a decision must be made as to how and how often this copy will be updated. If a pull mechanism is used, then the Sensor Management domain would have to poll the Track Management domain for changes in track data and retrieve those changes. If a push mechanism is used, then the Track Management domain would have to send track data changes to the Sensor Management domain, which would then have to apply these changes to its copy of the track data. Replication introduces performance issues (as this is yet another process that would have to be performed in the system) as well as data contention issues (as track data would probably have to be locked as this process is performed).

If this is the option selected, the design of the model problem would not have to change beyond what was expressed earlier, but then the functionality for updating track data, which is

outside of measurement-to-track association, would have to be added to the appropriate domain(s).

D.3 Summary

The problem of allocating functionality to domains is a part of Object-Oriented Analysis. In this appendix we have discussed an alternative for the allocation of measurement-to-track-association, as well as pros and cons of different approaches. The bottom line, we find, is that the allocation of functionality to domains is not only a function of the functional model, but also a function of non-functional models such as performance.

References

All URLs are valid as of the publication date of this report.

[Booch 99] Booch, G.; Jacobson, I.; & Rumbaugh, J. *The Unified Software Development Process.* Boston, MA: Addison-Wesley, 1999.

[Cockburn 00] Cockburn, A. *Writing Effective Use Cases.* Boston, MA: Addison-Wesley, 2000.

[Kennedy Carter 02] Kennedy Carter, Ltd. *iUML User Guide*. Surrey, UK: 2002.

[Mellor 02] Mellor, S. & Balcer, M. *Executable UML: A Foundation for Model-Driven Architecture.* Boston, MA: Addison-Wesley, 2002.

[ODMG 98] *ODMG OQL User Manual, Release 5.0.* Westboro, MA: Ardent Software, 1998. <http://www.cis.upenn.edu/~cis550/oql.pdf> (1998).

[OMG 03] *OMG Unified Modeling Language Specification, Version 1.5.* Needham, MA: Object Management Group, 2003. <http://www.omg.org/technology/documents/formal/uml.htm> (2003).

[Shlaer 92] Shlaer, S. & Mellor, S. *Object Lifecycles: Modeling the World in States*. Yourdon Press Computing Series. Englewood Cliffs, NJ: Prentice-Hall International, 1992.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

6. AGENCY USE ONLY (leave blank)	7. REPORT DATE	8. REPORT TYPE AND DATES COVERED
	September 2003	Final

9. TITLE AND SUBTITLE	10. FUNDING NUMBERS
A Model Problem Approach to Measurement-to-Track Association	C — F19628-00-C-0003
11. AUTHOR(S)	
Grace A. Lewis and B. Craig Meyers	

12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	13. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2003-TR-020
14. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	15. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2003-020

16. SUPPLEMENTARY NOTES

12.a DISTRIBUTION/AVAILABILITY STATEMENT	12.b DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (maximum 200 words)

This is the first in a series of reports that illustrate the use of model problems in the design of a system. The problem considered is measurement-to-track association. A “track” represents the state data about an object in the environment, and has a set of associated attributes. “Measurement-to-track association” is the process of determining the relation between a measurement and an existing track. In this process, tracks that meet particular attribute criteria can be selected via filters. This report examines the development and application of filters that can be used as selector mechanisms. The report also presents an initial design of the model problem, by using concepts and constructs from Unified Modeling Language (UML), executable UML (xUML), and Object-Oriented Analysis (OOA). Also covered are possible extensions to this work, related to performance considerations, additional filter types, and the distribution of filter information.

14. SUBJECT TERMS	15. NUMBER OF PAGES
acquisition; acquisition process; formal model	84
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL