Chapter 5
How Do You Detect an Asteroid That Might Hit the Earth?

This chapter pulls together a few pieces of the LSST model to tell the story of how the LSST software will identify moving objects from its torrent of images.

**Domain Model**

Let’s start with a small fragment from the LSST Domain Model, focusing on “Astronomical Objects” (AstroObjects). The domain model, in addition to it’s large main “almost everything” diagram, also is organized into subset diagrams showing clusters of related objects. It’s often a lot easier to look at the subset diagrams, and that’s the case here.

You can see from Figure 1 that we’ve identified a Solar System Object as a distinct kind of AstroObject, and that Solar System Objects have Orbits. We’ve also identified something mysterious called Ephemerides. As it turns out, Ephemerides are important to our story. But... (as I asked Tim on a recent visit to Tucson) what’s an Ephemeride?

![Figure 1. Domain model showing different kinds of Astronomical Objects.](image)

Here’s where Tim having defined his domain objects comes in handy. We can just look it up under E for Ephemeride. Using the domain model as a glossary is useful on any project, but when you have a distributed team like LSST, it’s absolutely essential (see Figure 2).
Figure 2. Using the Domain Model as a Project Glossary is a very useful technique.

As you can see, an Ephemeride is a Time Series of predicted Sky Coordinates for a Solar System Object. In other words, the predicted track of an object in the solar system (like a Martian Spacecraft, or an asteroid that might be heading for Earth, or the planet Neptune, or one of Saturn’s moons, or even the Hubble Space Telescope). In the words of an old electromagnetic theory professor of mine, this is "intuitively obvious to the casual observer."

**Modeling Tip: It all starts with Domain Modeling**

In complex systems like LSST, the Domain Model can become quite large. When this happens, create "subset view" diagrams that show a relate set of objects. Don’t forget to define all your domain classes unambiguously.

So far, so good. The LSST Domain Model has an object that exists specifically for the purpose we have in mind. Now let's take a look at how we're going to compute these Ephemerides.
Image Processing Pipelines

LSST’s image processing software uses a “pipeline” architecture. Images go in one end of the pipeline through an Input Queue, and are analyzed as they pass through various “processing stages”, then exit through an Output Queue. LSST’s middleware defines a general purpose architecture for pipelines which allows for parallel processing of the image stream. Parallel processing is an absolute necessity when you’re dealing with a stream of 3 gigapixel images with a new image coming through every few minutes. We’re going to be looking at one of many LSST pipelines later in this chapter, a pipeline called “Day MOPS”.

![Diagram of LSST's middleware managing image processing pipelines.]

Figure 3. LSST’s middleware manages the image processing pipelines.

Policies

LSST’s software will operate at much too high a rate for there to be human guidance and direction during the execution of a pipeline. However, there are many occasions where human guidance is necessary. LSST pipelines can be controlled by Policies, which are sets of parameters that human experts (astrophysicists) can define. So a Policy is really like a proxy object that replaces a person who would be guiding the image processing software if you slowed down the processing by a couple of million times. (See Figure 4).
3.2.2.6.1 Policy

Policy is a container for holding hierarchical configuration data in memory. A policy is a set of named parameters that can be used to configure the internal data and behavior of an object within an application.

An important feature of Policy objects is that the parameters can be loaded in from a file. Thus, it allows applications fine-grained control of objects even if much of the configuration parameters they provide are normally set to defaults and otherwise do not change. The Policy interface allows an application to pull out parameter values by name. Typically, the application "knows" the names it needs from a Policy to configure itself—that is, these names and the use of their values are hard-coded into the application. The application simply calls one of the get methods to retrieve a typed value for the parameter. (Nevertheless, if necessary, the parameter names contained in a policy can be retrieved via the \c names() member function.)

Figure 4. LSST’s pipelines are “policy driven”. Policy objects are sets of parameters that effectively act as proxies for expert astrophysicists. These parameters guide the image analysis, since the image processing runs far too quickly to allow for actual human intervention.

The Day MOPS Pipeline

Since LSST Pipelines are used to define workflow (and are not really use cases) we’re using an activity diagram to describe the workflow, instead of trying to describe the pipeline detail on a use case diagram. So it’s clear in the model that when we see an activity diagram, we’re working at the upper levels of the model.

Figure 5 shows the activity diagram that describes our moving object detection pipeline.

Figure 5. The Day MOPS pipeline detects moving objects and updates several of LSST’s Catalogs, including the Solar System Object Catalog.
Near Earth Object Detection

LSST's Day MOPS pipeline has been developed in a previous collaboration with a project called PAN-STARRS (Panoramic Survey Telescope & Rapid Response System). The PAN-STARRS website has an interesting discussion about Near Earth Object (NEO) detection and the potential threat from asteroids. 

Also, NASA JPL maintains a Near Earth Object website if you’d like to take a peek at which asteroids will be making close passes to us in the near future. The PAN-STARRS website lists their digital cameras as the world’s largest, at 1.4 gigapixels; this won’t be true anymore after LSST’s 3.2 gigapixel (that’s 3200 megapixels) camera gets built.

A Look Inside the Pipeline Stages

Within each activity, we’re elaborating the processing on a robustness diagram as if the activity were a use case, and we’re representing Policy as an actor, and (to make our robustness diagram rules work correctly) defining Policy Readers as boundary objects. Since the robustness diagram is a conceptual design diagram, it doesn’t really matter whether a policy reader object will ever be implemented. We’re using it in the model as a device to help us identify what the different policy parameters should be.

Modeling Tip: Robustness diagrams help “Object Discovery”

Robustness diagrams (aka “Martian”) are conceptual design diagrams that are useful to help you discover details about your object model. Since it’s not an implementation model, it’s OK to add conceptual objects like Policy Readers to help us discover what data we’re reading from the Policy. See Figure 6 for an example of a robustness diagram.

By elaborating pipeline stages within activities, and allowing for “controllers” (lower level algorithms) within the robustness diagram, we’re putting a limit on how many levels of “algorithms within algorithms within algorithms within algorithms” we’re going to show in the model. The controllers that are connected to the Policy Readers help us to identify exactly what needs to go into the Policy.

Modeling Tip: Avoid deep trees of nested functions

One of the problems with functional decomposition approaches that are typically used in algorithm-intensive systems is that there tend to be many levels of nested algorithms. ICONIX Process for Algorithms represents high-level (workflow) algorithms on activity diagrams, elaborates policy/parameter-driven activities on robustness and sequence diagrams, and lower-level “number cruncher” algorithms as controllers within the robustness diagrams.

So it’s much easier to figure out what level of algorithm you’re looking at when you read the model.

---

1 http://pan-starrs.ifa.hawaii.edu/public/
2 http://pan-starrs.ifa.hawaii.edu/public/asteroid-threat/asteroid_threat.html
3 http://neo.jpl.nasa.gov/
4 http://pan-starrs.ifa.hawaii.edu/public/design-features/cameras.html
5 http://www.lsst.org/lsst/gallery/camera/suzanne
Notice that even though we’re using it to describe a policy-directed algorithm instead of a use case, the robustness diagram still works. This allows us to leverage other capabilities of ICONIX Process and the Sparx Agile/ICONIX add-in to generate skeleton sequence diagrams and test cases automatically.

**Modeling Tip: ICONIX/Algorithms retains all the benefits of ICONIX Process**

The standard, use-case-driven, ICONIX Process discovers details about an object model using robustness analysis, and does a responsibility-driven allocation of behavior using sequence diagrams. Additionally, the Agile/ICONIX add-in from Sparx Systems supports automatic generation of test cases and JUnit/NUnit test code from robustness and sequence diagrams. ICONIX Process for Algorithms retains all of these benefits.

ICONIX Process for Algorithms is useful for a wide range of algorithm-intensive systems.
Figure 7 shows an algorithm for a pipeline stage that hasn't been elaborated (yet) on a robustness diagram. Exactly as with use-case-driven ICONIX Process, drawing the robustness diagram helps us discover additional domain objects that may still be missing from the model. So our process tailoring has retained the benefits of use-case-driven ICONIX Process, without trying to force-fit algorithms into use cases.

Figure 7. This algorithm for computing Ephemerides will soon be detailed on a robustness diagram.
Converging Towards a Solution

You can see the benefits of a forward-modeling and reverse-engineering approach here, with a reverse-engineered database schema for MOPS (see Figure 8).

By using LSST’s R&D Phase for both modeling and prototyping different areas of functionality, the project minimizes risk. When LSST reaches the Construction phase, there will be very little doubt of a successful outcome.

**Modeling Tip: Modeling + Reverse Engineering minimizes risk**

Once again, you can see how the capabilities of the modeling tool (in this case, EA’s ability to reverse-engineer database schemas in addition to a wide range of programming languages) allows for a risk-mitigation strategy that’s crucial for a project of the complexity of LSST.

**Conclusion**

Detecting asteroids which might impact the Earth is a small, but important, portion of LSST’s overall science mission. Hopefully, you’ve been able to follow the discussion presented in this chapter that explains how this capability will be provided by the Day MOPS image processing pipeline. If you have, that’s a good sign that our tailoring of ICONIX Process has resulted in a UML model that successfully and unambiguously explains the design of the LSST software.