## Designing the Classes

Once a set of candidate objects is determined… we must:

Determine which are "real" objects in the system.

Identify their attributes.

- attributes are data
- define what the data is, not how it is to be represented (that comes later)

Identify their responsibilities.

- public services (behaviors) the object must provide
- may imply certain attributes necessary to provide those services
- define what the service is, not how it to be accomplished
- some services may be private, but those are usually identified later
- services are invoked though message passing

## Identifying Attributes

An attribute is a single characteristic which is common to all instances of a class.

Look for adjectives and possessive phrases in the requirements document.

Find a general description of the object.

Determine what parts of the description are applicable to the problem domain.

Four categories of attributes:

- descriptive
- naming
- state information
- referential (relationship links)

## Eliminating Attributes

Some apparent attributes may be considered independently of the objects — make those objects in their own right.

- Rumbaugh: if an attribute is changed in the system w/o being part of any entity, then it should be an object.

Relationships among objects may also have attributes. Do not confuse those with attributes of the involved objects.

Eliminate minor details that do not affect methods.

## Specifying Attributes

An attribute should be atomic (simple).

Eliminate attributes that can be calculated from others.

Eliminate attributes that address normalization, performance, or other implementation issues.

Verify that the attributes make semantic sense together.

## Data Versus State

| Data | State |
|---|---|
| <u>Definition:</u> Information processed by the system | <u>Definition:</u> Information used by system to control processing |
| <u>Examples from Minor 1:</u> a record offset; a GIS record | <u>Examples from Minor 1:</u> type of current command |

## Identifying Responsibilities

Look for verb in the requirements document — usually this will define services of the object of the sentence

E.g. Quarterback throws the ball.

This defines a service <u>for</u> the ball, <u>provided by</u> the quarterback.

Look at user scenarios — different ways the system components can be used.

Look at each feature — require services of many objects.

## Specifying Responsibilities

Name the service to match the external request for the service.

- reportFID()
- serveNextCommand()
- getRecordAtOffset()

Identify the information and/or entities necessary to provide the service.

- GIS record object
- command file, command file processor

Identify the responses, if any, that the service will generate.

- feature ID (cannot fail unless object not initialized)
- no more commands in file
- invalid file offset

## Example: File Navigation Project

Consider the GISRecord class:

Name: GISRecord

Attributes:

FeatureID

FeatureName

. . .

(or just a single String object?)

Responsibilities:

Report FeatureID

Provide displayable representation

. . .

**Further questions:**

**When are the attributes set?**

**Which of the attributes are mutable?**

## Example: File Navigation Project

Consider the CommandParser class:

| |
|---|
| Name: CommandParser |
| Attributes:<br>RandomAccessFile (on commands file) |
| Responsibilities:<br>Report next command, if any<br>Transformation of command from raw form to internal form<br>. . . |

## Example: File Navigation Project

Consider the CommandProcessor class:

Name: CommandProcessor

Attributes:

FileWriter (on log file)

(assoc to) GISRecordFileParser object

Responsibilities:

Determine command type

Carry out command

. . .

## Guidelines for Designing the Classes

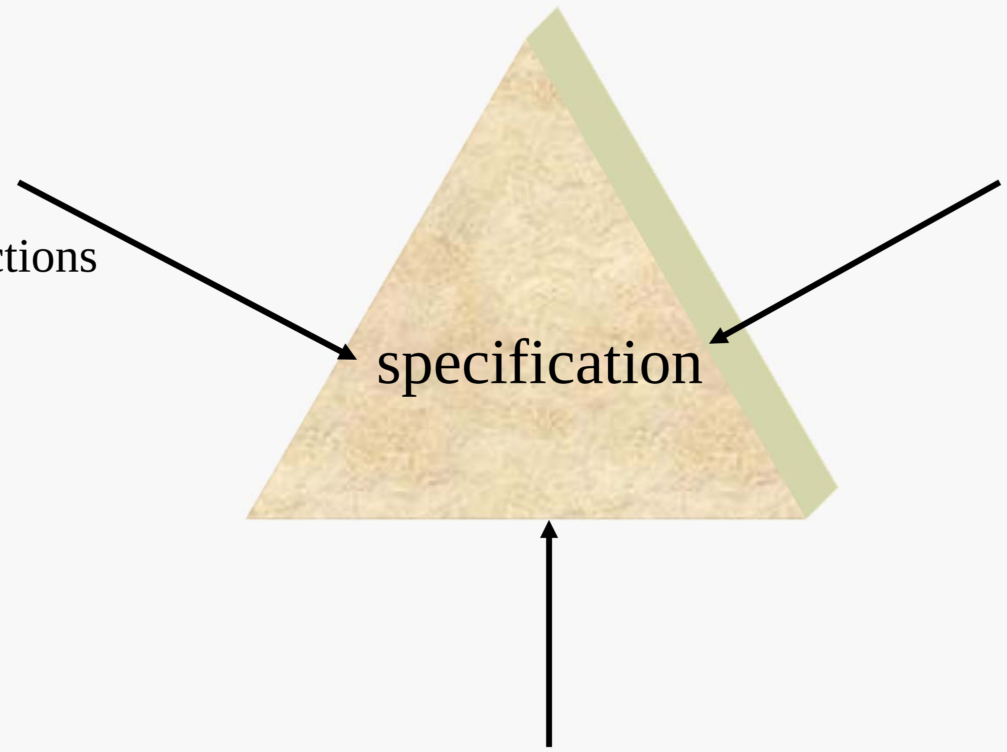
We need a systematic way of determining the attributes and responsibilities of a class.

Otherwise, we run a large risk of missing essential features.

To identify attributes and responsibilities the designer must ask the right questions regarding the system being designed.

We can provide some guidance in choosing what questions to ask…

## Design Perspectives



Behavioral

Emphasizes actions in system

Informational

Emphasizes role of information/data/state and how it’s manipulated

Structural

Emphasizes relationships among components

Kafura

## Example: File Navigation Project

Behavioral (actions):

- **file offsets** of **GIS records** are reported (by who?)
- **GIS records** are retrieved from the data file (by who?)

Structural (relationships):

- **GISRecordFileParser** knows about the GIS record file
- **CommandParser** knows about the command file
- **CommandProcessor** knows about the **GISRecordFileParser**
- **Controller** knows about the **CommandParser** and the **CommandProcessor**

Informational (state):

- a **Command** may be record_at/exit/??

## Behavioral Perspective

Consider some action in a program…

What object...

- initiates action?

What objects...

- help perform action?
- are changed by action?
- are interrogated during action?

Consider retrieving a GIS record…

**CommandProcessor**...

- initiates the action

**GISRecordFileParser**…

- performs the action

No objects or state information…

- are changed* by the action

Patron List…

- is interrogated during the action

## Behavioral Categories

| | |
|---|---|
| Actor | (does something, typically initiates) |
| Controller | |
| Reactor | (system events, external & user events) |
| Controller, CommandProcessor (?) | |
| Agent | (messenger, server, finder, communicator) |
| possibly CommandParser, GISRecordFileparser | |
| Transformer | (data formatter, data filter) |
| possible CommandParser, GISRecordFileParser | |

## Structural Perspective

What objects...

- are involved in relationship?
- are necessary to sustain (implement, realize, maintain) relationship?

What objects not in relationship...

- are aware of and exploit relationship?

Consider a relationship: CommandProcessor knows GISRecordParser

Controller…

- is involved in establishing the relationship

??…

- is necessary to sustain the relationship

Controller...

- is aware of and exploits the relationship

## Structural Categories

Acquaintance (symmetric, asymmetric)

- Controller knows about CommandProcessor, asymmetric relationship

Containment (collaborator, controller)

- GISRecordFileParser controls/uses RandomAccessFile
- similar issue with CommandParser

Collection (peer, iterator, coordinator)

- Controller knows and manages CommandParser and CommandProcessor
- no data structrures issues as yet, but they would qualify

## Informational Perspective

What objects...

- represent the data or state?
- read data or interrogate state?
- write data or update state?

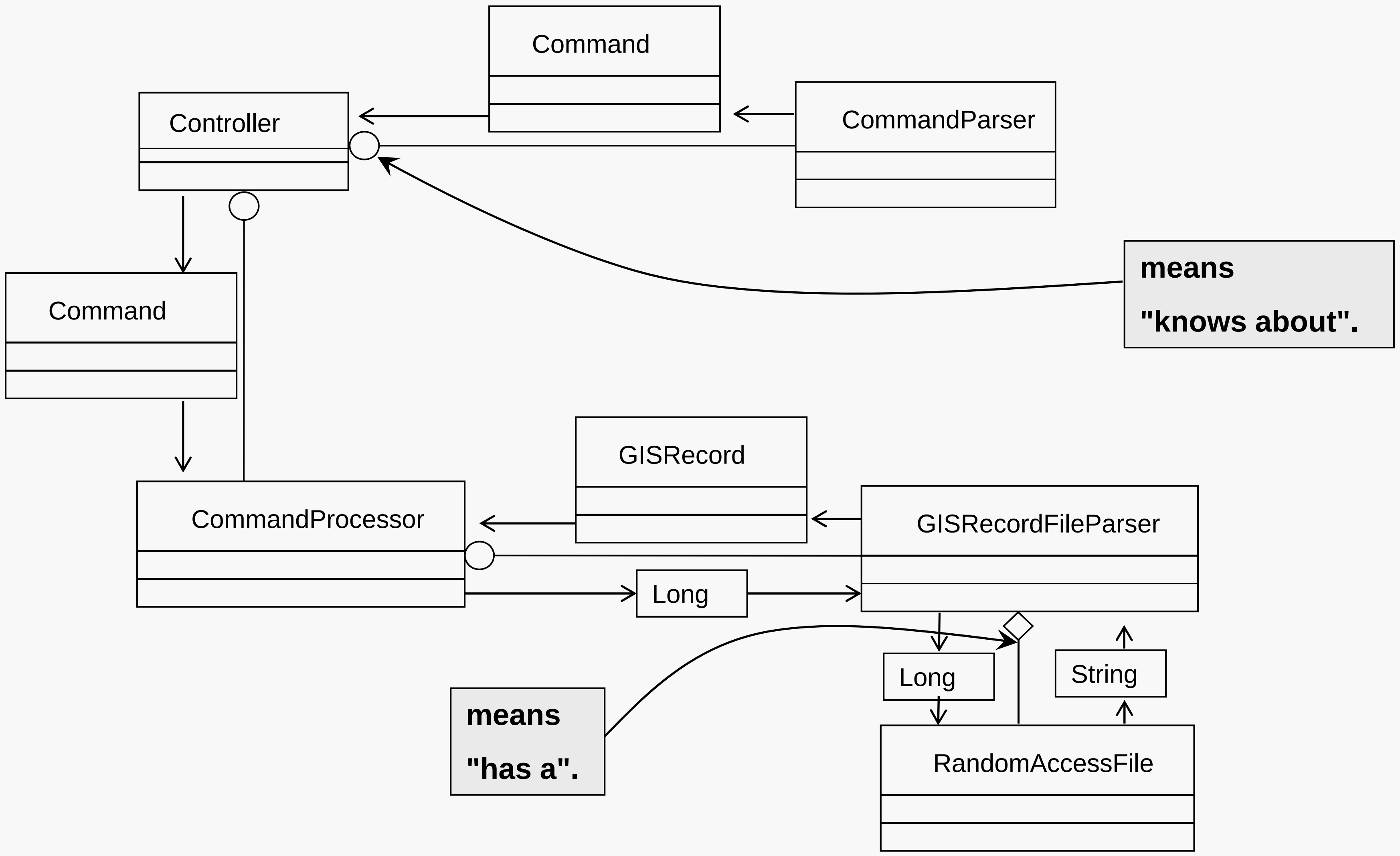
Consider a state:  type of current command

CommandParser and/or CommandProcessor…

- represent (stores) the state information

## Example: Preliminary Overall Design

Here's a partial, preliminary design, based on the preceding discussions:


Command
Controller
CommandParser
Command
means "knows about".
GISRecord
CommandProcessor
GISRecordFileParser
Long
Long
String
means "has a".
RandomAccessFile