Vince:

- Initial assignment should be vars (with propagation to specialization), but there aren’t any in the libs.
- Vars “propagate” to other features. Discard when we didn’t get to bidirectional var/nonvar combination (see emails with Ed).
- Var/nonvar
    - Features on nonoccs become nonvars on occ that “aggregate” vars.
        - Some lib readonlies might just as well be nonvars.
        - Some nonvars (eg, steps) might
    - Nonvars on occs accompanied by vars
        - Eg, steps (nonvar) and currentSteps (var). Likely apply to “root” features.
- Ends
    - See previous.
    - Might be able to move Link readonlies to LinkObject.

Conrad:

- All occ features are vars, except the exceptions (which are alot).

Joint:

- Not likely to finish varying the libs.
- Not good to do it partially bc people will think the unmarked ones are intentionally nonvars.
- Validation of taxonomic between vars and nonvars unexamined.

1. Features of non (don’t specialize, aren’t disjoint with) occurrence classes

Note: *Features on nonoccurrence classes that should be vars when specialized to occurrences (including "top-level" features) require a* ***constraint on Occurrence to make them vars****.*

1.1. Readonly features

Must be vars when specialized to occurrence classes (unless they're intended to be features only of lives).

```
  Links::Link:participant:Anything[2..*] nonunique ordered;
  Links::BinaryLink::source/target: Anything[1] nonunique subsets participant;
    Another data point for the JIRA discussion on participants as (non) vars.

  Clocks::universalClock:UniversalClockLife[1]
  Observation::defaultMonitor[1] : DefaultMonitorLife
  SpatialFrames::feature defaultFrame : DefaultFrameLife[1]
    Above are domain Anything ("top level").
    Must be const to prevent deleting the (single) *Life, then
     immediate recreating and "resetting" feature (logically
      possible).
```

1.2. "Root, top-level" features on non-occurrence classes

All intended to "change" over time when specialized to occurrence classes.

```
  Base::things: Anything [1..*] nonunique
  Base::dataValues: DataValue[0..*] nonunique subsets things
  Base::naturals: ScalarValues::Natural[0..*] subsets dataValues

  //Can these change dynamically?
  Base::zeroOrOne [0..1]  (multiplicity)
  Base::oneToMany [1..*]  (multiplicity)
  Base::zeroToMany [0..*] (multiplicity)

  Links::links: Link[0..*] nonunique subsets things
  Links::binaryLinks: BinaryLink[0..*] nonunique subsets links
  Links::selfLinks: SelfLink[0..*] nonunique subsets binaryLinks
```

Other features on non-occurrence classes that should be vars when specialized to occurrences

```
  Not
    Base::Anything::self: Anything[1] subsets things chains things.that
    Base::things::that : Anything[1]
    Links::*
```

2. Features of occurrence classes

2.1. Readonly features

Are these intended to be features only of lives?

```
 Transfers::Transfer:isInstant/isMove/isPush: Boolean[1]
  StatePerformances::StatePerformance:incomingTransitionTrigger : MessageTransfer [0..1]
default null
```

2.2. "Root" features on occurrence classes

```
  Metaobjects::feature metaobjects : Metaobject[0..*] :> objects
  Objects::objects: Object[0..*] nonunique subsets occurrences {
  Objects::linkObjects: LinkObject[0..*] nonunique subsets links, objects intersects
links, objects {
  Objects::binaryLinkObjects: BinaryLinkObject[0..*] nonunique subsets binaryLinks,
linkObjects

  Performances::, Transfers:: top-level features, all for steps,
    exprs, transfers, flows.  I figure these are non-vars (at least
    for steps, transfers, and flows, since their specializations will
    be linked by successions).
```

2.3. Other features on occurrence classes that should be vars

Features specializing vars must be vars (omitted here).

```
   //Can these change dynamically?
    Metaobjects::Metaobject::annotatedElement:Element[1..*]
    Metaobjects::SemanticMetadata::redefines annotatedElement : Type[1]
```

```
Occurrences::occurrences: Occurrence[0..*] nonunique subsets things;
Occurrences::earlierFirstIncomingTransferSort : IncomingTransferSort

Clocks::Clock::currentTime:NumericalValue[1]

Occurrences::Occurrence::suboccurrence
  Root for composition semantics.

Occurrences::Occurrence:: (purely) Space relations between occurrences
  These'll typically change over time, tho they can also relate things
    happening at separate times (ie, aren't always time-aligned).


??  (Might be const)
  Occurrences::incomingTransfersToSelf subsets incomingTransfers {
  Occurrences::Occurrence::outgoingTransfersFromSelf subsets outgoingTransfers
  Occurrences::Occurrence::incomingTransferSort
  Occurrences::Occurrence::isDispatch : Boolean[1] default false
  Occurrences::Occurrence::dispatchScope: Occurrence [1] default self;
  Occurrences::Occurrence::runToCompletionScope: Occurrence [1] default self;

Objects::Object::involvingPerformances: Performance[0..*] subsets performances
  (specialized by enactedPerformances)
  Steps, but only bc they're typed by performances.  Linkable by successions?

LinkObjects

??
  "Root" Performances::Performance

??
  SemanticMetadata::baseType : Type[1]
  Occurrences::Occurrence::localClock:Clock[1] default universalClock
    Should snapshot always use the clock of their life?  If not, I
    guess this could be a var.
  Occurrences::Occurrence:incoming/outgoingTransfersTo/FromSelf

Not vars
  Occurrences::Occurrence::Time relations between occurrences
    Intended for the entire time of the occurrence
  Clocks::Clock::timeFlowConstraint
    Already written against snapshots (simpler as var)
  Observation::*
  SpatialFrames::*
```