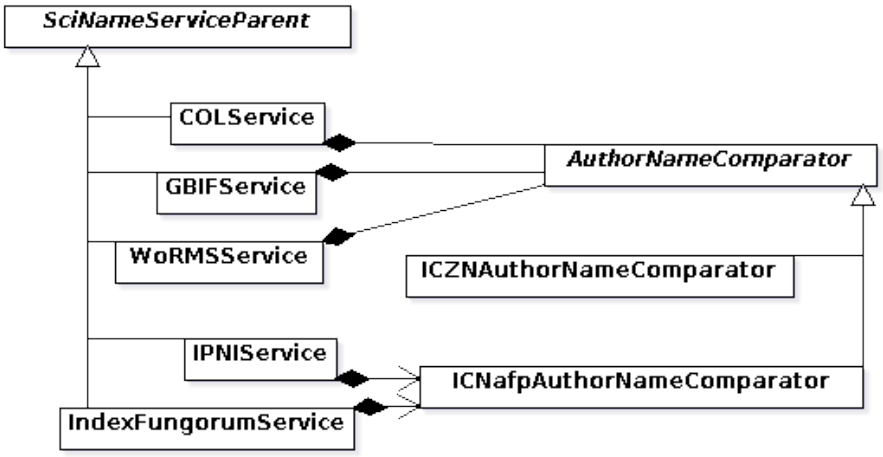
Kurator Markup Challenges for YesWorkflow

FP-Akka marks up the composition of actors in the workflow using YesWorkflow markup. Individual actors in FP-Akka have their internals delegated to classes in FP-KurationServices, where the implementation details tend to be in sets of object oriented classes. For example, SciNameServiceParent is an abstract class, with concrete IPNIService, GBIFService, WoRMSService, children composed with a concrete subclass of AuthorNameComparator. Following the logic of the data processing in the internals of the scientific name validation actor poses challenges for the dataflow oriented YesWorfklow markup.



Timothy McPhillips made the following notes in KURATOR-106 concerning challenges that markup of object oriented code provides for YesWorkflow (YW).
The first challenge is that actors calling these services operate on, update, or add specific fields to input records they receive. YW needs a way of identifying the fields in records and declaring that a code block works specifically with those fields. See KURATOR-153.
A second challenge is that the particular service classes used by an actor can depend on data values determined at run time. For example, NewScientificNameValidator recognizes multiple authorities (WORMS, COL, etc) and creates instances of the corresponding service classes (WoRMSService, COLService, etc) as needed. In YW terms, the subworkflow(s) invoked at run time depend on data values available at run time. One way of handling this is to mark up all possible subworkflows and allow YW graphs (and queries) to be simplified by asserting that particular subworkflows will be used and others not (see KURATOR-146). Alternatively, the different options can be represented in the YW graphs with a graphical indication that data is routed dynamically between different workflow branches (see KURATOR-141) depending on data values.

A third challenge is that some values produced by these services are derived incrementally. For example, multiple remote service can be called with each service name being appended to a growing string listing all of the services called for a particular record. Comments provided by the services are aggregated similarly. To accurately represent the provenance of these values YW must be able to recognize data values that are incrementally produced, with multiple steps contributing parts of the values, rather than produced all at once. The declarations of fine-grained provenance relationships need to handle such cases (KURATOR-149).

A fourth challenge is the conditional control-flow constructs within the services. For example, a name validation service operating on a record may or not find a matching name at the corresponding authority. What fields are updated, and how, depends on such run time conditions. YW needs to provide a way of representing alternative data flows within a single (sub)workflow. See KURATOR-141.

A fifth challenge is that code invoked in the processing of a single record can span multiple functions declared at a number of different levels of one or more class hierarchies. YW needs annotations that can declare that a function is called within a code block, with that function declared via YW in the same class, in a parent class, or in a completely different class used by the calling class or function. See KURATOR-161, KURATOR-159, and KURATOR-145.