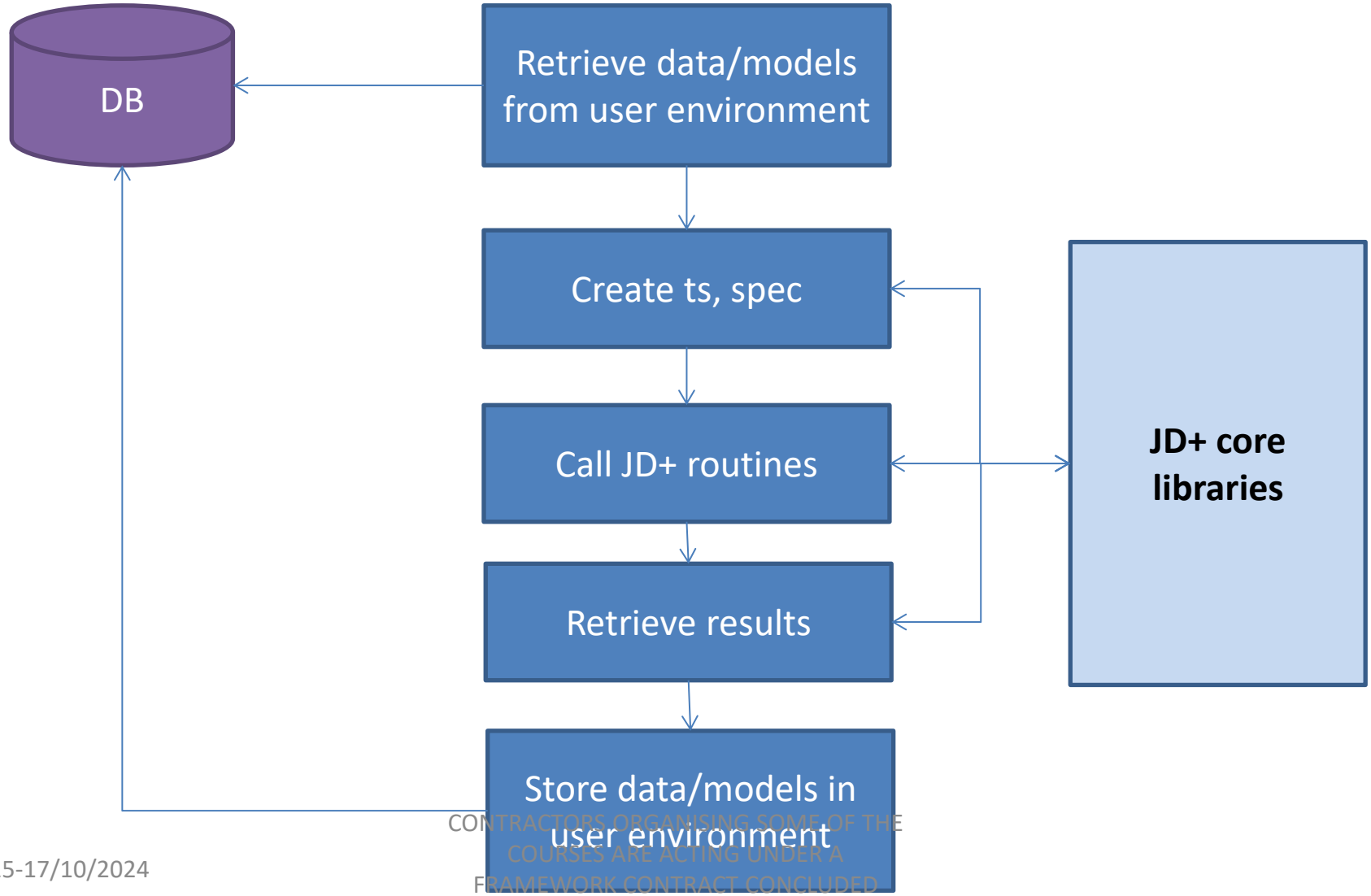
Routine SA processing with JD+

ESTP Training

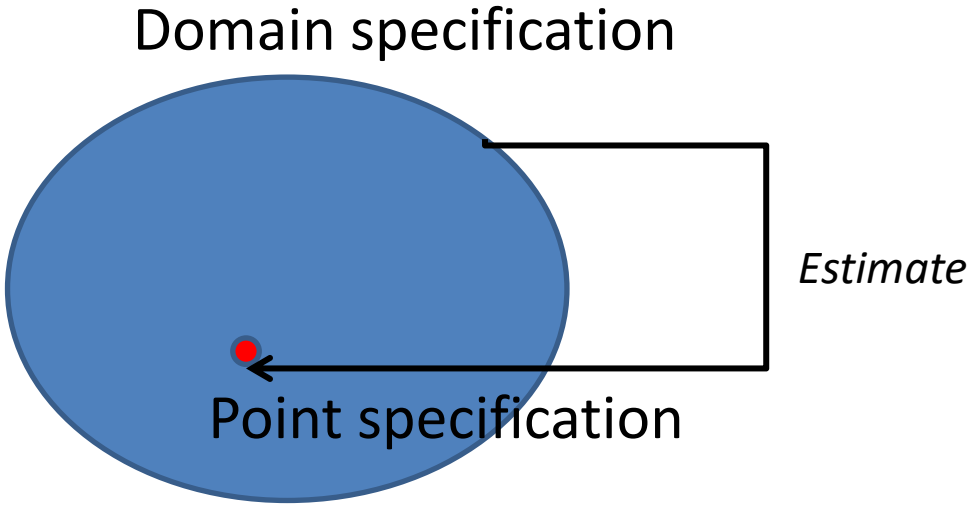
In-house developments

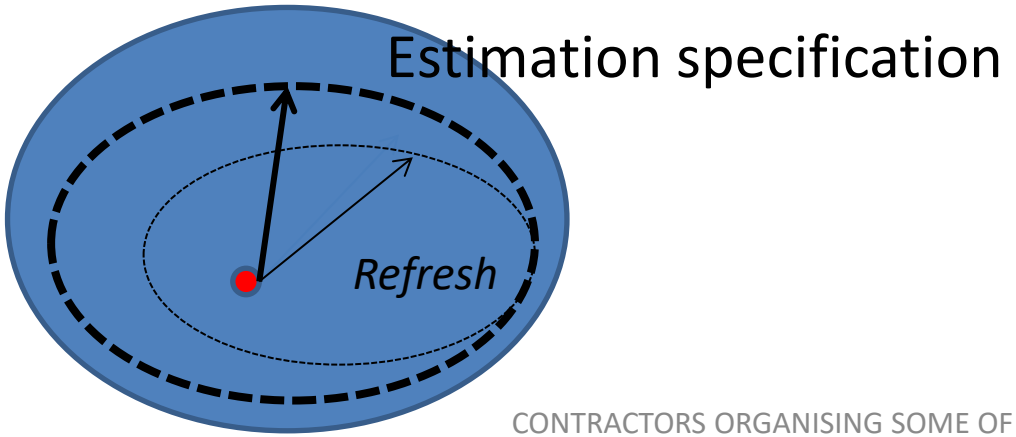


Java code

```java
// time series
Ts s=createTs();
//item, with a default specification
SaItem item = createItem(s);

// compute (results stored in the item
CompositeResults rslt = item.process();

// Generic data retrieval through the "getData" method
// we can retrieve any result using keys used - for instance - in the cruncher (see the WIKI of the cruncher)
SarimaModel arima = rslt.getData("arima", SarimaModel.class);
TsData sa = rslt.getData("sa", TsData.class);

System.out.println(arima);
System.out.println(sa);

// Advanced data retrieval
SeatsResults seats=rslt.get("decomposition", SeatsResults.class);

UcarimaModel ucm = seats.getUcarimaModel();
System.out.println(ucm);

// refreshing the specification corresponding to the estimated model ('point specification')
// null could be replaced by a frozen domain (outliers not modified for that span)
// last param to remove pre-specified time span for the series (seldom used)
TramoSeatsSpecification newSpecification = (TramoSeatsSpecification) TRAMOSEATS.createSpecification(item, null, EstimationPolicyType.FreeParameters,

// That new specification can be used for further processing
// for instance
// In practice, s should have been updated with new obs. The estimation policy type is just for information (no actual impact on the computation)
SaItem nitem=item.newSpecification(s, newSpecification, EstimationPolicyType.FreeParameters);
CompositeResults nrslt = nitem.process();

TsData nsa = nrslt.getData("sa", TsData.class);
```

JD+ solution. Principles



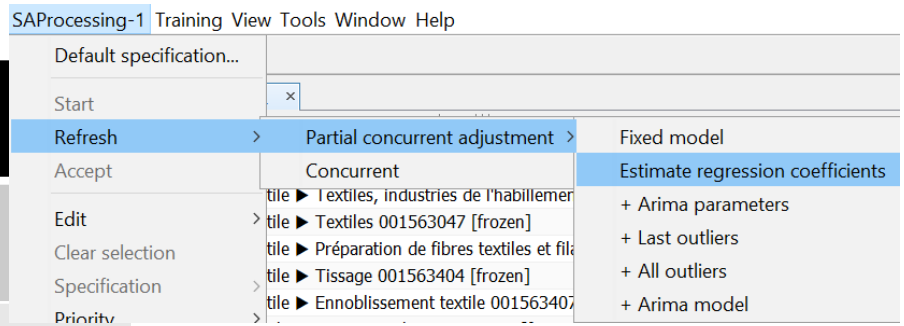


- Define the initial(or domain) specification
- The estimation gives the point specification (=the result)
- Refreshing the model « relaxes » some constraints; the new specification (=estimation specification) is always inside the domain specification

- Tip: keep the initial specification as large as possible

Updating policies

Policy	Re-estimation (cumulative)
Fixed model (partial concurrent)	Nothing
Regression coefficients (partial concurrent)	Regression coefficients
Parameters (partial concurrent)	Coefficients of the ARIMA model
Last outliers (partial concurrent)	Outliers of the last year
Outliers (partial concurrent)	All outliers
Arima (partial concurrent)	ARIMA orders
Concurrent	Log/level + calendar effects



Updating policies

- Why?
  - We want to minimize the revisions (at least in the short term).
  - Origins of the revisions (decreasing importance)
    - Structure of the model (outliers, ARIMA…)
    - Parameters of the ARIMA model (Seats!)
    - Coefficients of the regression variables
    - News (forecasts ≠ actual data) →use forecasted seasonal factors or AO (fixed for revisions, free for new observations)

JD+ GUI (I)

- Suitable for sets < 3000 series
- Steps
  - Initialization
    - Define the default specification (or use an existing one)
    - Modify some specifications (keep them « large »)
    - Generate and export the results
    - Save the processing

JD+ GUI (II)

- Next estimations
  - Re-load the workspace
  - Refresh the processing (for instance)
    - For normal re-estimation, refresh only the parameters
    - Refresh the model once each year (or each two years)
    - Inspect the results and modify them if need be (will modify some « domain specifications »
  - Exports the results and save the workspace.

JWSACruncher (I)

- Suitable for any number of series (split the workspace is several SAProcessing) if need be
- Steps
  - Initialization:
    - generate the input file (=workspace), using the GUI (don’t process the multi-processing) or using in-house programs

JWSACruncher (II)

- Define the setting file for the cruncher
  - Output, format, paths…

```
▼<wsaConfig bundle="10000" csvlayout="list" csvseparator="," ndec="6">
   <policy>parameters</policy>
  ▶<matrix>...</matrix>
  ▶<tsmatrix>...</tsmatrix>
 </wsaConfig>
```

- Call the command line

– Next estimations

- Modify the setting file (if need be)
- Re-call the command line

RJDemetra3

```r
library(rjd3toolkit)
library(rjd3tramoseats)

#partial series
s<-window(retail$BookStores, end=2009)

#execute tramoseats
q<-tramoseats(s, "rsafull")

#define a frozen domain and refresh the recent outliers
new_spec<-tramoseats.refresh(q$result_spec, policy='Outliers', end='2006-12-31')

# re-execute with the refreshed specification and new data (here tge complete series)
newq<-tramoseats(retail$BookStores, new_spec)

# the specifications can be stored in DB or as R-objects
```