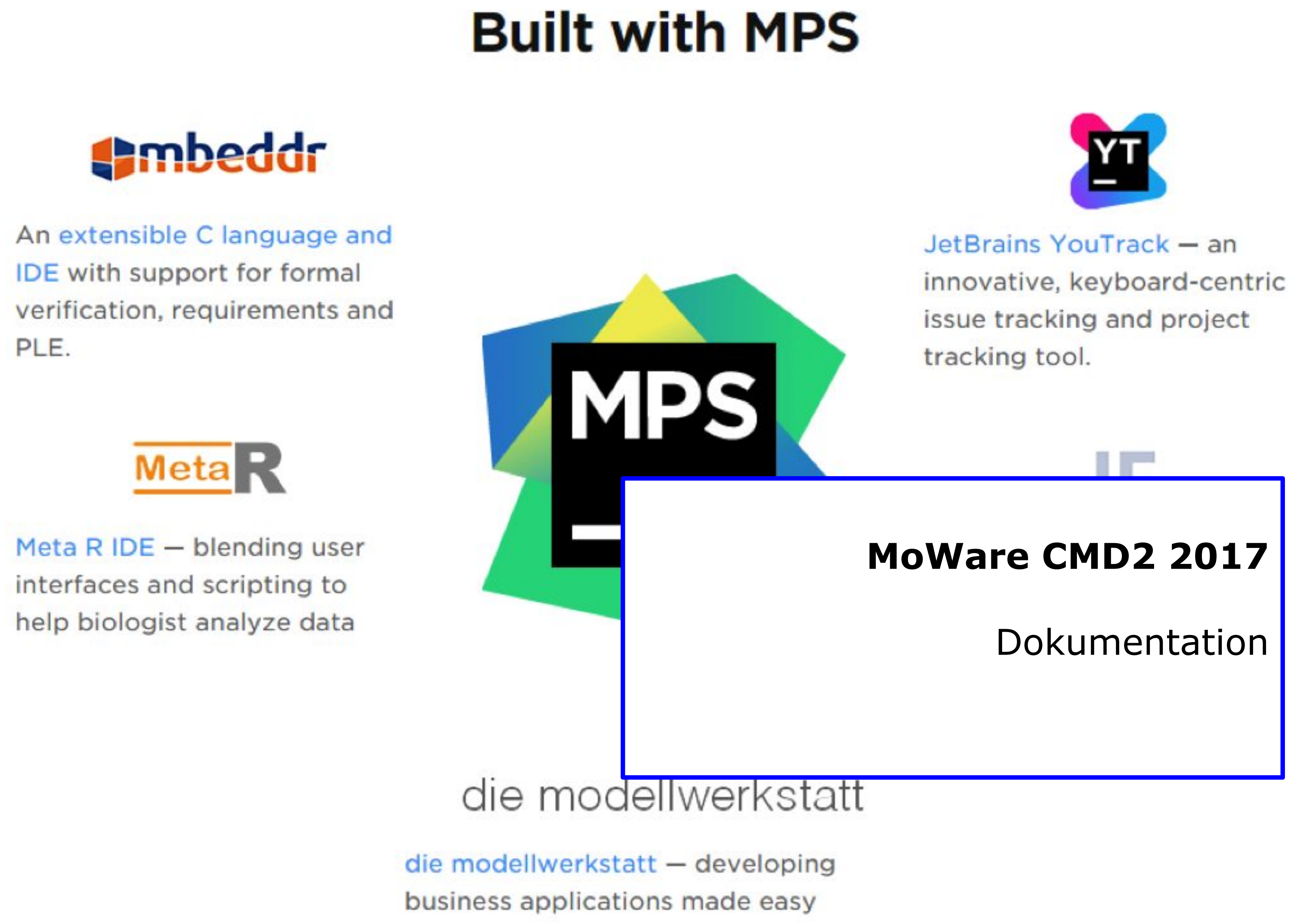
MoWare CMD2 2017

Dokumentation



Built with MPS

An extensible C language and IDE with support for formal verification, requirements and PLE.

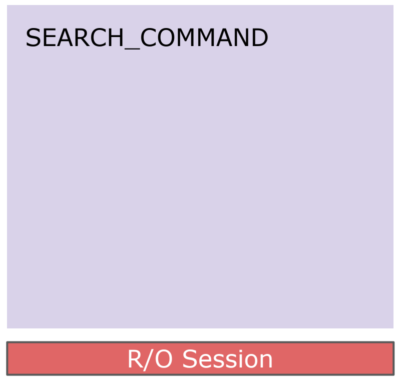
JetBrains YouTrack — an innovative, keyboard-centric issue tracking and project tracking tool.

Meta R IDE — blending user interfaces and scripting to help biologist analyze data

die modellwerkstatt — developing business applications made easy

MoWare CMD 2017

Command Types Moware CMD RC38

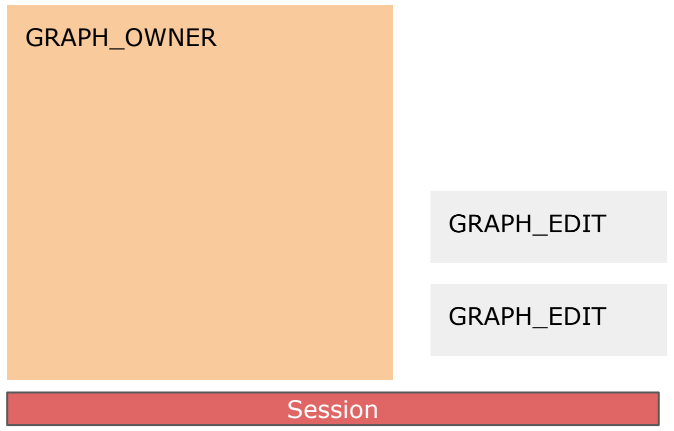


Search Command to look for entities

> can not modify entities
> load entities in read-only mode
> session can not start transactions

> filter-search pattern as only applicable pattern
  Page 1: User enters search criteria
  Page 2: User receives result-list

> Internal session-keystores are cleared before
  Any page-init occurs *[allow for reload functionality]*



Graph_Owner Command to modify graph of entities

> Graph_Owner provides a read/writeable session with transaction
> Checkout entities and assemble them to a graph
> Visualize this graph within a page of the Graph_Owner
> Provide a "Save & Close" Button to let the user save the current graph

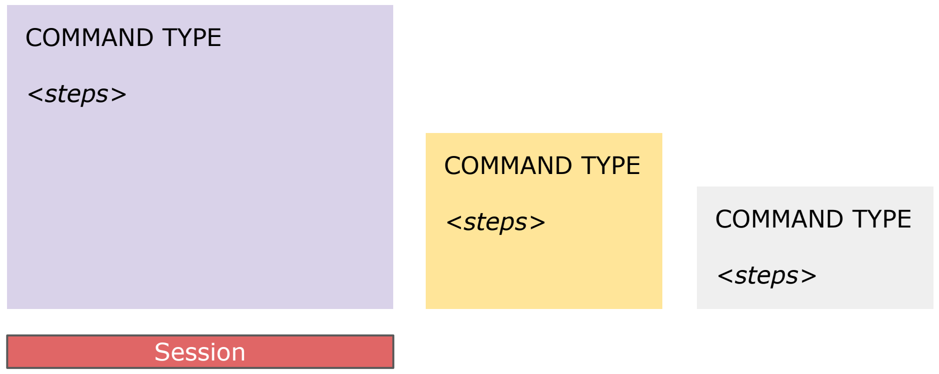
> Use Graph_Edit commands to edit the graph directly
> Only one Graph_Edit (modal prompt window)
> Shared session between Graph_Edits and Graph_Owner

Summary command types

> **SEARCH_COMMAND**
Read only session, keystores are cleared on every page-init

> **GRAPH_OWNER + GRAPH_EDIT**
GRAPH_OWNER comes with a read/write session
Only one GRAPH_EDIT can be executed at the same time

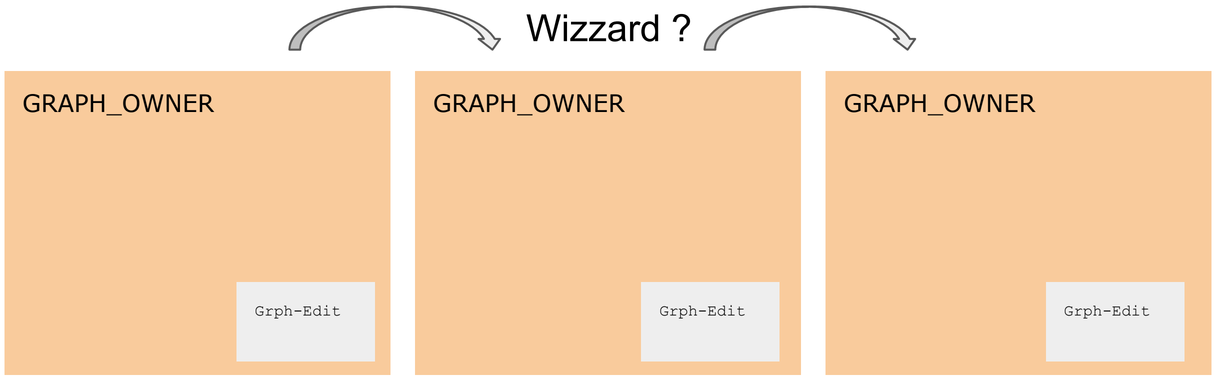
> **GRAPH_OWNER_MODAL**
Behaves exactly like a GRAPH_OWNER, except that users are prevented from switching to other tabs before terminating the GRAPH_OWNER_MODAL



Command Patterns

- ***Do not repeat yourself in code*** (command flexibility, minimize num of commands, enhance maintainability)
- Agree on a common application structure, a common style on how to handle recurring requirements
- Repeatable common solution for all apps
- Articulate app situations more clearly, to allow refacts in future
- Division of concerns: separate domains for reusage
- Allow Top-Level testing of command sequences
- MoWare Support for patterns arrangements are well tested

The concept: Command in Sequenz ausführen



```
OpenXava supports getNextModule()
Module END, RUN getNextModule()
MoWare:
GRAPH_OWNER -> GRAPH_OWNER -> GRAPH_OWNER ??
```

1. SP - Standard Pattern
2. W - Wizzard Pattern
3. UD - Update Pattern
4. FSP - Filter Search Pattern
5. MEP1 - Multiple Execution Pattern (GRAPH_OWNER)
6. MEP2 - Multiple Execution Pattern (GRAPH_EDIT)
7. PP - Print Pattern (aka Status Change Pattern)
8. SGO - Sub Graph Owner Pattern
9. CEP - Create Edit Pattern
10. THP - Task Handling Pattern
11. GCP - Graph Composition Pattern
12. BEP - Base Entity Pattern
13. BJH - Batch Job Pattern

Document Centered Applications

Document with Information
Modelled as a Graph

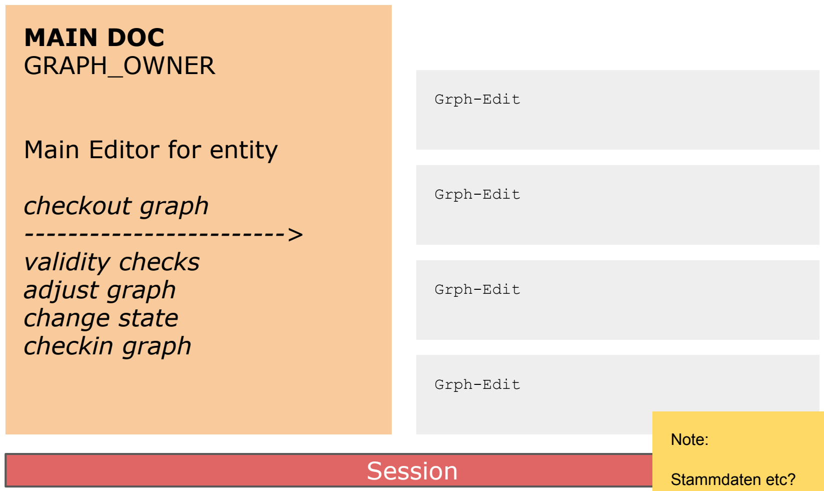
Checkout the Document - give me the document
Checkin the Document - give the doc back
Version / Unit of Work / Transaction

<u>ACID paradigm</u>
Atomicity - all or nothing
Consistency - one state to the next
Isolation - concurrent execution
Durability -fully persisted

1. Pass KEY of document to GRAPH_OWNER
2. Checkout document completely
3. <Adjust Graph according to Business Logic>
4. <Validate Graph according to Business Logic>

5. (let user edit the document)
6. <Adjust Graph according to Business Logic>
7. <Validate Graph according to Business Logic>
8. Checkin document completely

9. Forward some information from document to application (in order to update states)



Note:

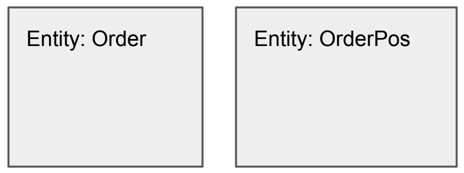
Stammdaten etc?
Changing Artikel Info during Day?
Scopes etc.

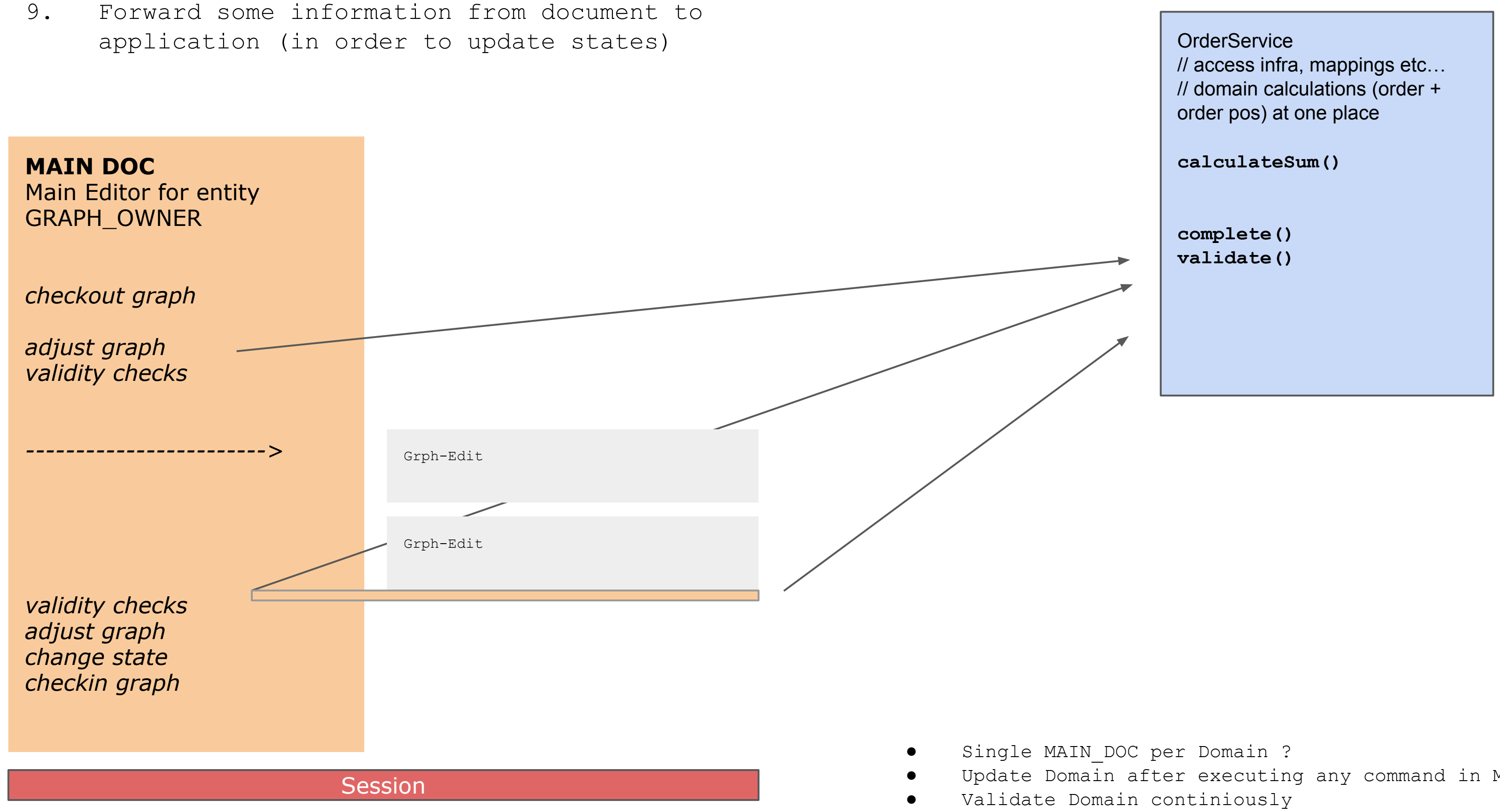
Document Centered Applications

1. Pass KEY of document to GRAPH_OWNER
2. Checkout document completely
3. <Adjust Graph according to Business Logic>
4. <Validate Graph according to Business Logic>

5. (let user edit the document)
6. <Adjust Graph according to Business Logic>
7. <Validate Graph according to Business Logic>
8. Checkin document completely

9. Forward some information from document to application (in order to update states)



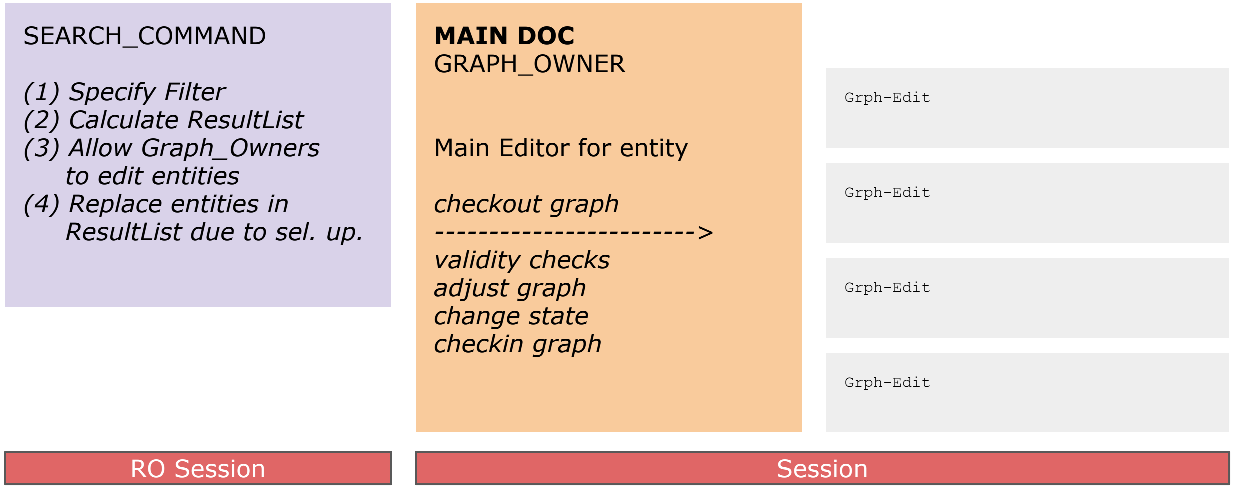


- Single MAIN_DOC per Domain ?
- Update Domain after executing any command in MAIN_DOC
- Validate Domain continiously
- Never deviate from this concept (sub-go?)

SP - Standard Pattern

Edit Documents of a certain type - The**MAIN_DOC GRAPH_OWNER**

> Handles Session + checkout / checkin of graph
> Has a single Page
> Uses cancel, Flag and Conditions
> Must be available almost always! At least in RO mode!



SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

MAIN DOC
GRAPH_OWNER

Main Editor for entity

checkout graph
------------------------->
validity checks
adjust graph
change state
checkin graph

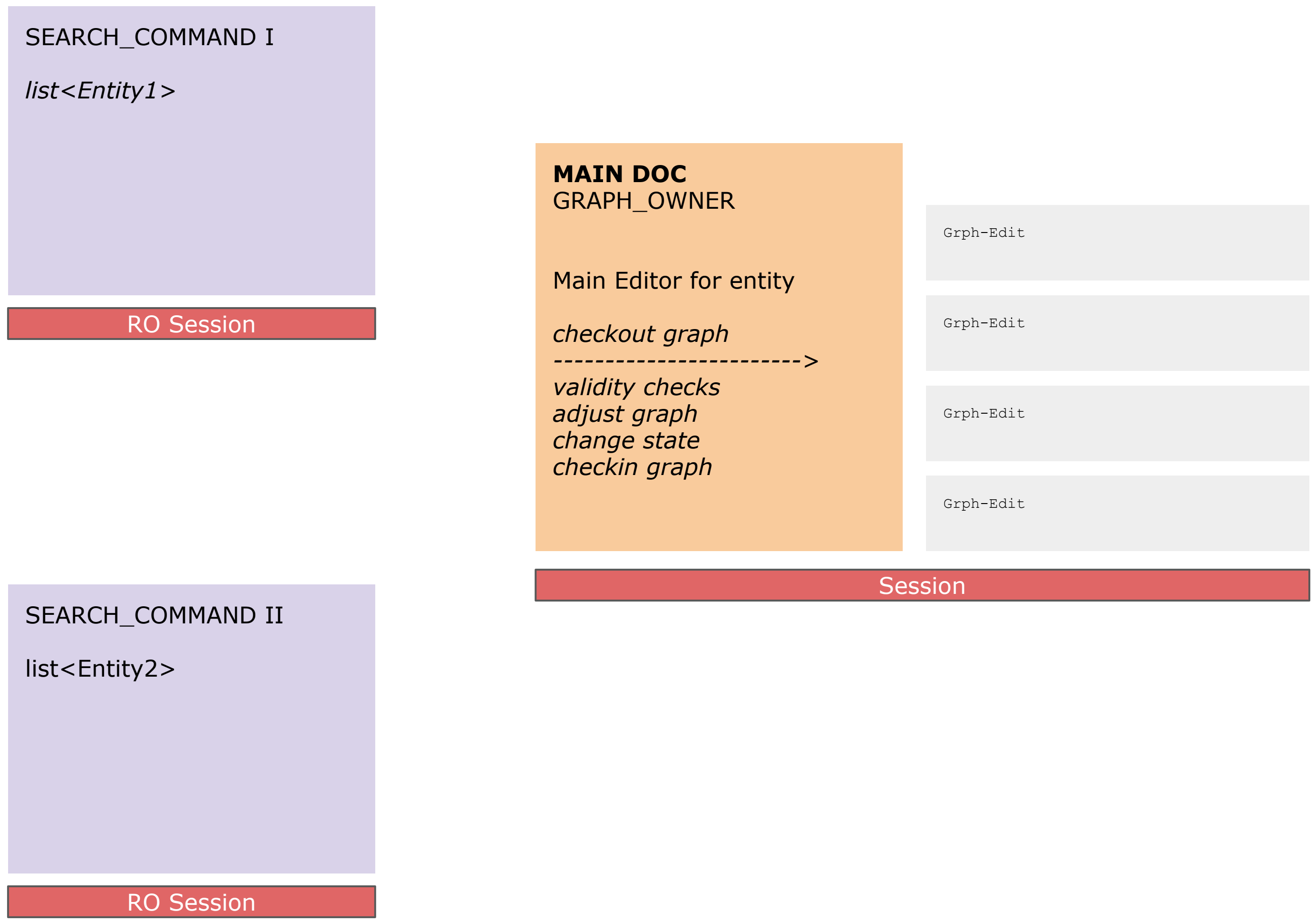
(1) Checkout Graph
(2) Adjust & Validate Graph
(3) ***<User can edit>***
(4) Adjust & Validate Graph
(5) Checkin Graph

??? Right now, we can not come up with validation checks during page init / command init to warn user on problems with graph?????

Right now the assumption is that all existing graphs are valid, were validated!

SP - Standard Pattern (SUOP)

```
The Selection Update On Parent

> Goal: Update information in search-result after main-doc execution
> Replace complete entity in search commands graph
> Throw away old entity, assign entity from main-doc
> New: Multiple selection updates possible for differentSEARCH CMDs
```

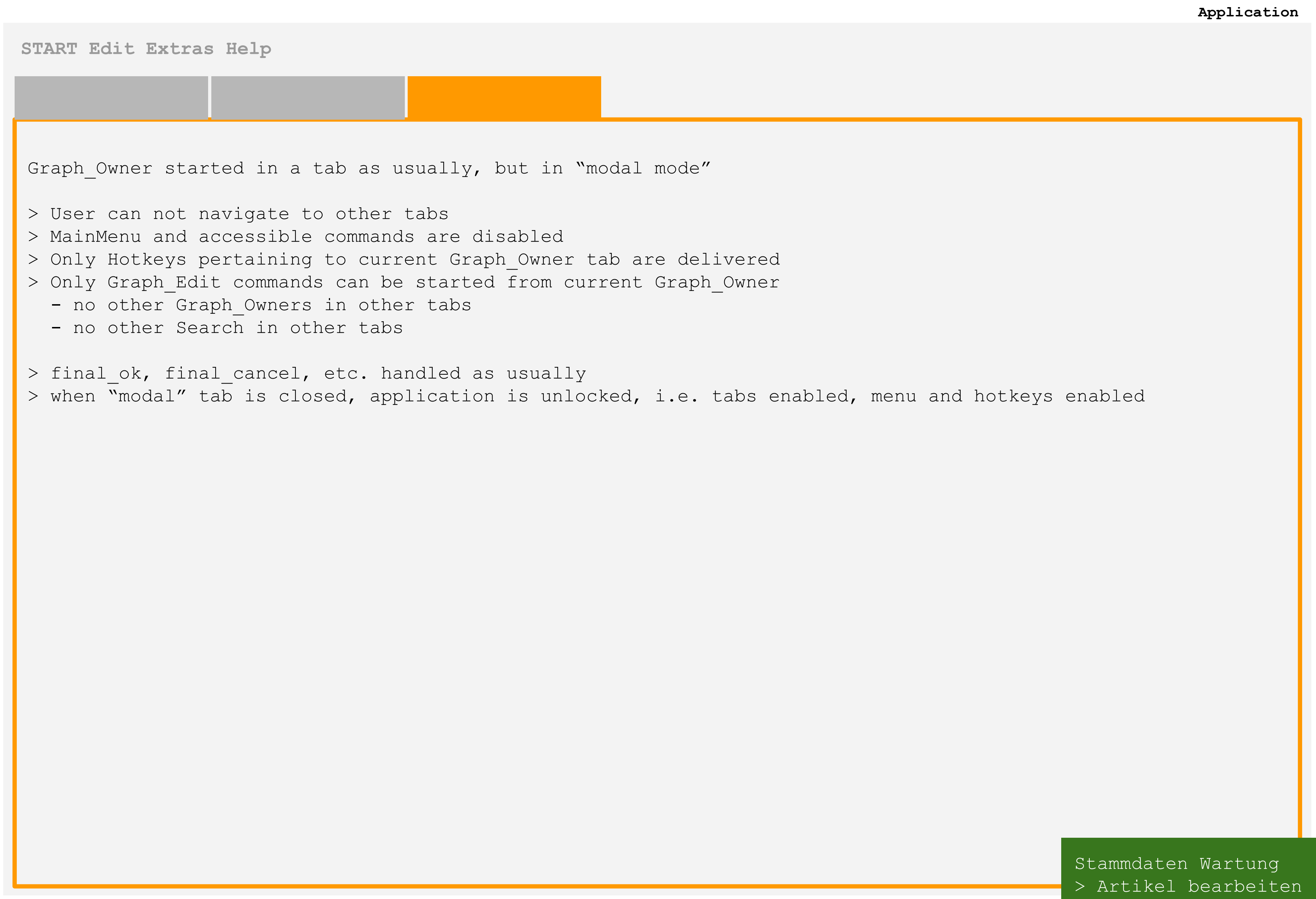


MAIN_DOC Graph Owner

```
class RekoAkt {

public RekoAkt flattenGraph(void) {

     this.referenz = null;

     this.list.clear();
     this.list = null;

     return this;

}

}
```

Command selection update: akt.flattenGraph()

Standard Pattern: Graph_Owner in modal mode



Graph_Owner started in a tab as usually, but in "modal mode"

```
> User can not navigate to other tabs
> MainMenu and accessible commands are disabled
> Only Hotkeys pertaining to current Graph_Owner tab are delivered
> Only Graph_Edit commands can be started from current Graph_Owner
  - no other Graph_Owners in other tabs
  - no other Search in other tabs

> final_ok, final_cancel, etc. handled as usually
> when "modal" tab is closed, application is unlocked, i.e. tabs enabled, menu and hotkeys enabled
```

Stammdaten Wartung
> Artikel bearbeiten

FSP - Filter Search Pattern

```
Search Entities with user given filter

Page1: Filter DelegateForm
Page2: Result Table
```

ViewObject MyFilterVO
Filter Property 1
Filter Property 2
Filter Property 3
+ ResultList

SEARCH_COMMAND

(1) Set <default> values for all filter properties
(2) User can adjust values of filter properties
(2) Calc result and assign MyFilterVo.resultList
(3) Graph_Owner might edit result entities
(4) Graph_Owner issues selection-updates for Search_Command Page2 (bound to MyFilterFo.resultList)

Page1 Filter DelegateForm

***boundTo:* MyFilterVO**

Delegate 1
Delegate 2
Delegate 3
…
…

Page2: Result Table

boundTo:* MyFilterVO*.resultList

Entity 1
Entity 2
Entity 3

RO Session

!

```
Do not pass entities across sessions ! (pass id only)
Do not issue selection-updates with huge object graphs !
?
```

```
Nach Dokumenten suchen
> Akte suchen
```

W - Wizzard pattern

```
Wizzard to enrich information across multpile pages

Page 1: UI entry
Page 2: again UI entry
Page 3: more UI entry / editing
```

GRAPH_OWNER / GRAPH_EDIT

(1) one single command handles multiple UI

(?) division of concerns? Does the data structures and pages relate strongly to each other?
(?) Jumps forward/backward are possible, but is it necessary?
(?) railway oriented programming - cancel / done stops whole command

Page1 UI entry

***boundTo:* Graph Entity**

Delegate 1
Delegate 2
Delegate 3
…
…

Page2 UI entry

***boundTo:* Graph Entity**

Delegate 1
Delegate 2
Delegate 3
…
…

Page3 UI entry

***boundTo:* Graph Entity**

Delegate 1
Delegate 2
Delegate 3
…
…

Session

```
Mehrstufig erfassen
> Complex Graph_Edit
```

UD - Update pattern

```
Use UPDATE hotkey to issue conclusions on "delegate leave"
```
> Do not issue this conclusion on ESC ?
> No business logic / code in UI, no java hooks

```
Page 1: UI entry
Page 2: again UI entry
..
```

GRAPH_OWNER / GRAPH_EDIT

Page1 UI entry

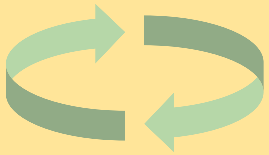
***boundTo:* Graph Entity**

Delegate 1
Delegate 2
Delegate 3
…
…

(1) page provides conclusion with hotkey UPDATE
(2) conclusion changes Graph of Entities and #Meta-Infos
(3) conclusion reloads page again, page-init is executed
(4) scopes are recalculated

PAGE_INIT

PAGE_CONCLUSION



Page2 UI entry

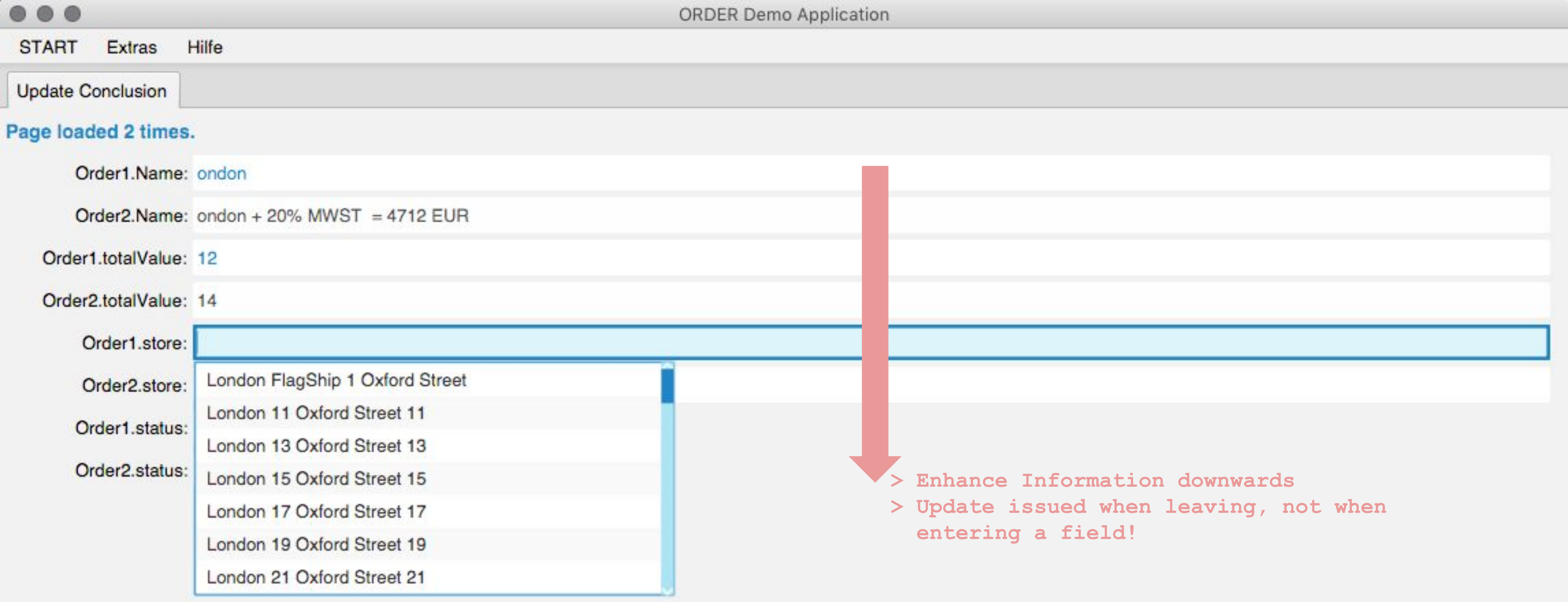
***boundTo:* Graph Entity**

Delegate 1
Delegate 2
Delegate 3
…
…

(1) page handles

Session

ReferenzDelegate: Artikel zu spezifischen Warengruppen



ORDER Demo Application

START Extras Hilfe

Update Conclusion

Page loaded 2 times.

Order1.Name: ondon

Order2.Name: ondon + 20% MWST = 4712 EUR

Order1.totalValue: 12

Order2.totalValue: 14

Order1.store:

Order2.store:

Order1.status:

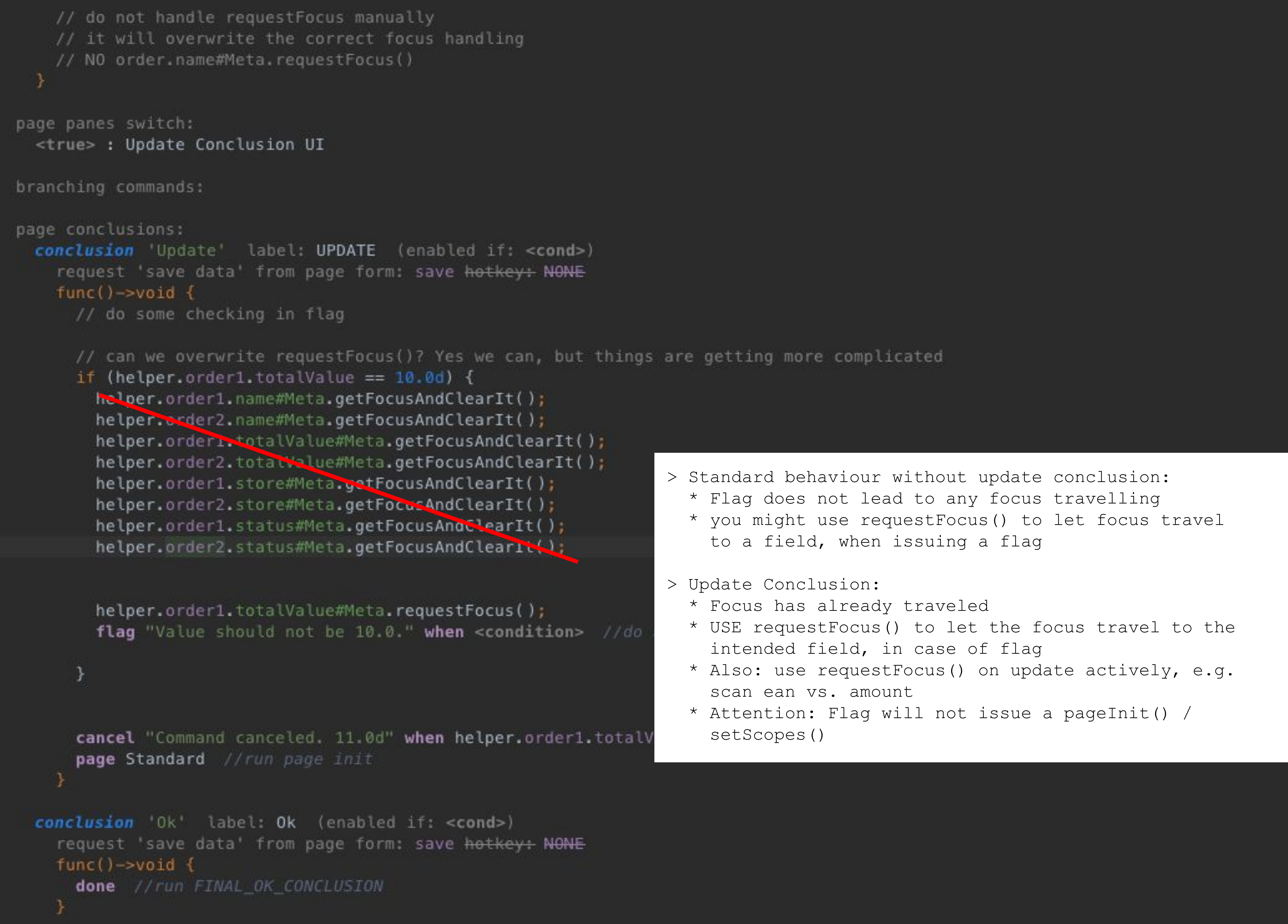
Order2.status:

London FlagShip 1 Oxford Street
London 11 Oxford Street 11
London 13 Oxford Street 13
London 15 Oxford Street 15
London 17 Oxford Street 17
London 19 Oxford Street 19
London 21 Oxford Street 21

> Enhance Information downwards
> Update issued when leaving, not when entering a field!

```
then, calc page title: "Page loaded " + pageLoadCnt + " times."

set scopes for page:
  pageSetScopesFunc()->void {
    // For the sake of consistency in a UI delegate form, all delegates have to issue the update conclusion
    // except the last one (no update necessary, select a conclusion to proceed)
    // all others have to issue the update to keep the UI in sync when traveling frwd/backward with the focus
    // or skipping a particular delegate !

    // (1) Reset Scopes for page
    helper.order1.store#Meta.setScope(allStores);
    helper.order2.store#Meta.setScope(allStores);
    helper.order1.status#Meta.setAllElements();
    helper.order2.status#Meta.setAllElements();


    // ------------------------------------------------------------------------
    if (!"".equals(helper.order1.name.trim(both))) {
      helper.order2.name = helper.order1.name + " + 20% MWST  = 4712 EUR";
    } else {
      helper.order2.name = "";
    }
    helper.order2.name#Meta.setEnabled(false);

    helper.order2.totalValue = helper.order1.totalValue * 1.2d;


    // (2) Set specific scope
    helper.order1.store#Meta.setScope(allStores.where({~it => it.name.contains(helper.order1.name); }).toList);
    // (3) Reset value, if not in scope
    if (!helper.order1.store#Meta.getScope().contains(helper.order1.store)) {
      // all dependent scopes have to be cleared.
      helper.order1.store = null;
    }



    // ------------------------------------------------------------------------
    // (2) Set specific scope
    if (helper.order1.store != null) {
      helper.order2.store#Meta.setScope(new arraylist<Store>{helper.order1.store});
      helper.order2.store = helper.order1.store;

    } else {
      // (3) Reset value if determinable
      helper.order2.store = null;
    }
```

```
> In update conclusion, or in setScopes()
> Assuming, user changed any of the fields
  * recheck all scopes and their values
  * reset values in case they are no longer valid!
  * adopt meta-info appropriately
```



```
    // do not handle requestFocus manually
    // it will overwrite the correct focus handling
    // NO order.name#Meta.requestFocus()
  }

page panes switch:
  <true> : Update Conclusion UI

branching commands:

page conclusions:
  conclusion 'Update'  label: UPDATE  (enabled if: <cond>)
    request 'save data' from page form: save ~~hotkey:~~ ~~NONE~~
    func()->void {
      // do some checking in flag

      // can we overwrite requestFocus()? Yes we can, but things are getting more complicated
      if (helper.order1.totalValue == 10.0d) {
        ~~helper.order1.name#Meta.getFocusAndClearIt();~~
        ~~helper.order2.name#Meta.getFocusAndClearIt();~~
        ~~helper.order1.totalValue#Meta.getFocusAndClearIt();~~
        ~~helper.order2.totalValue#Meta.getFocusAndClearIt();~~
        ~~helper.order1.store#Meta.getFocusAndClearIt();~~
        ~~helper.order2.store#Meta.getFocusAndClearIt();~~
        ~~helper.order1.status#Meta.getFocusAndClearIt();~~
        ~~helper.order2.status#Meta.getFocusAndClearIt();~~


        helper.order1.totalValue#Meta.requestFocus();
        flag "Value should not be 10.0." when <condition>  //do

      }


      cancel "Command canceled. 11.0d" when helper.order1.totalV
      page Standard  //run page init
    }

  conclusion 'Ok'  label: Ok  (enabled if: <cond>)
    request 'save data' from page form: save ~~hotkey:~~ ~~NONE~~
    func()->void {
      done  //run FINAL_OK_CONCLUSION
    }
```

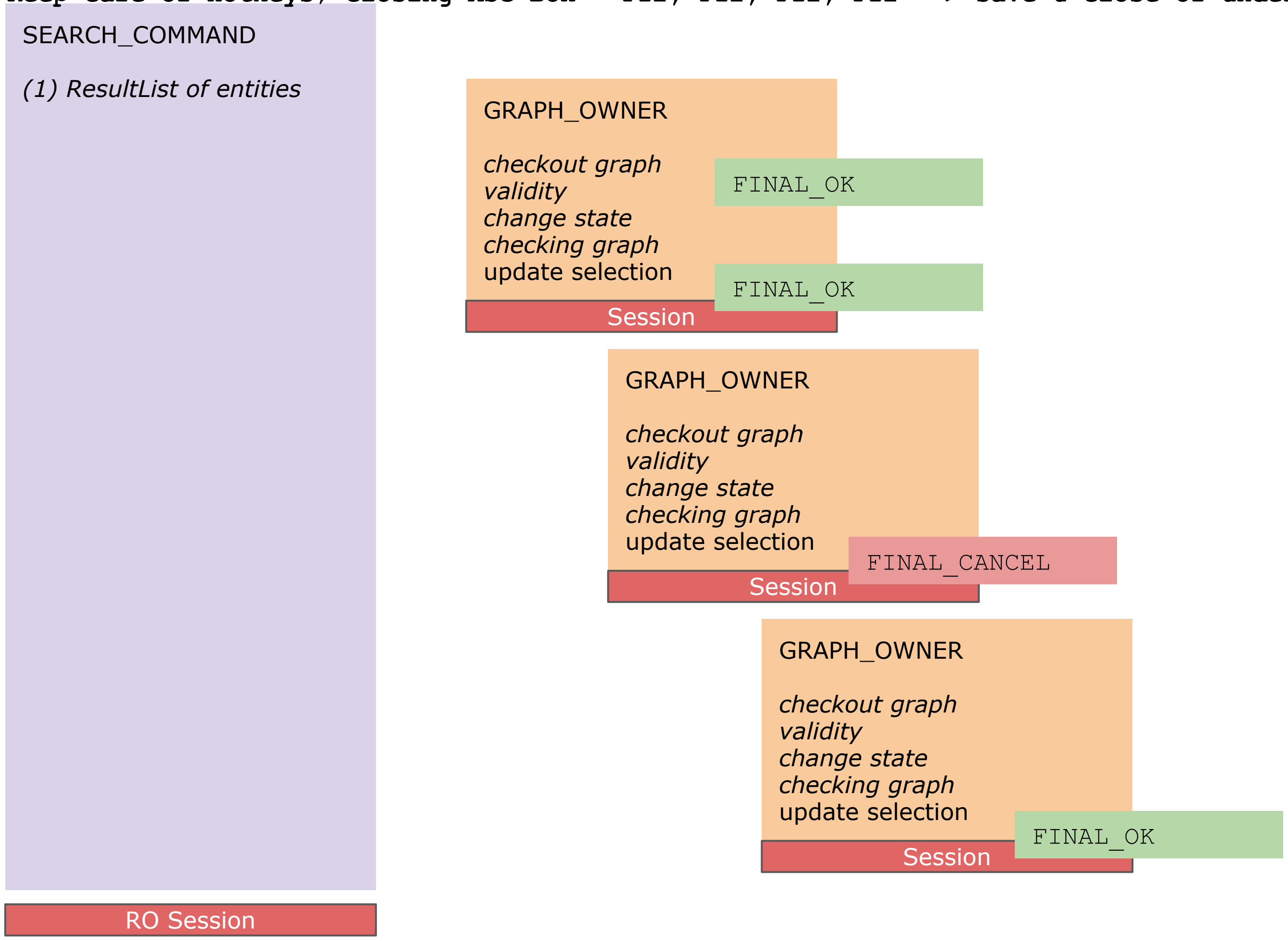
```
> Standard behaviour without update conclusion:
  * Flag does not lead to any focus travelling
  * you might use requestFocus() to let focus travel
    to a field, when issuing a flag

> Update Conclusion:
  * Focus has already traveled
  * USE requestFocus() to let the focus travel to the
    intended field, in case of flag
  * Also: use requestFocus() on update actively, e.g.
    scan ean vs. amount
  * Attention: Flag will not issue a pageInit() /
    setScopes()
```

MEP1 - Multiple Execution Pattern (GRAPH_OWNER)

Execute a command multiple times on a *selected list of entities*
without stopping the execution due to cancel BUT in case of an exception

Command has to provide a session (GRAPH_OWNER)
not requiring any user interaction (no page, auto-conclusion-mode)

Enabled condition of all Graph_Owners have to evaluate to true!
Keep care of hotkeys; closing MSG Box - F12, F12, F12, F12 - > save & close of underlying SEARCH?

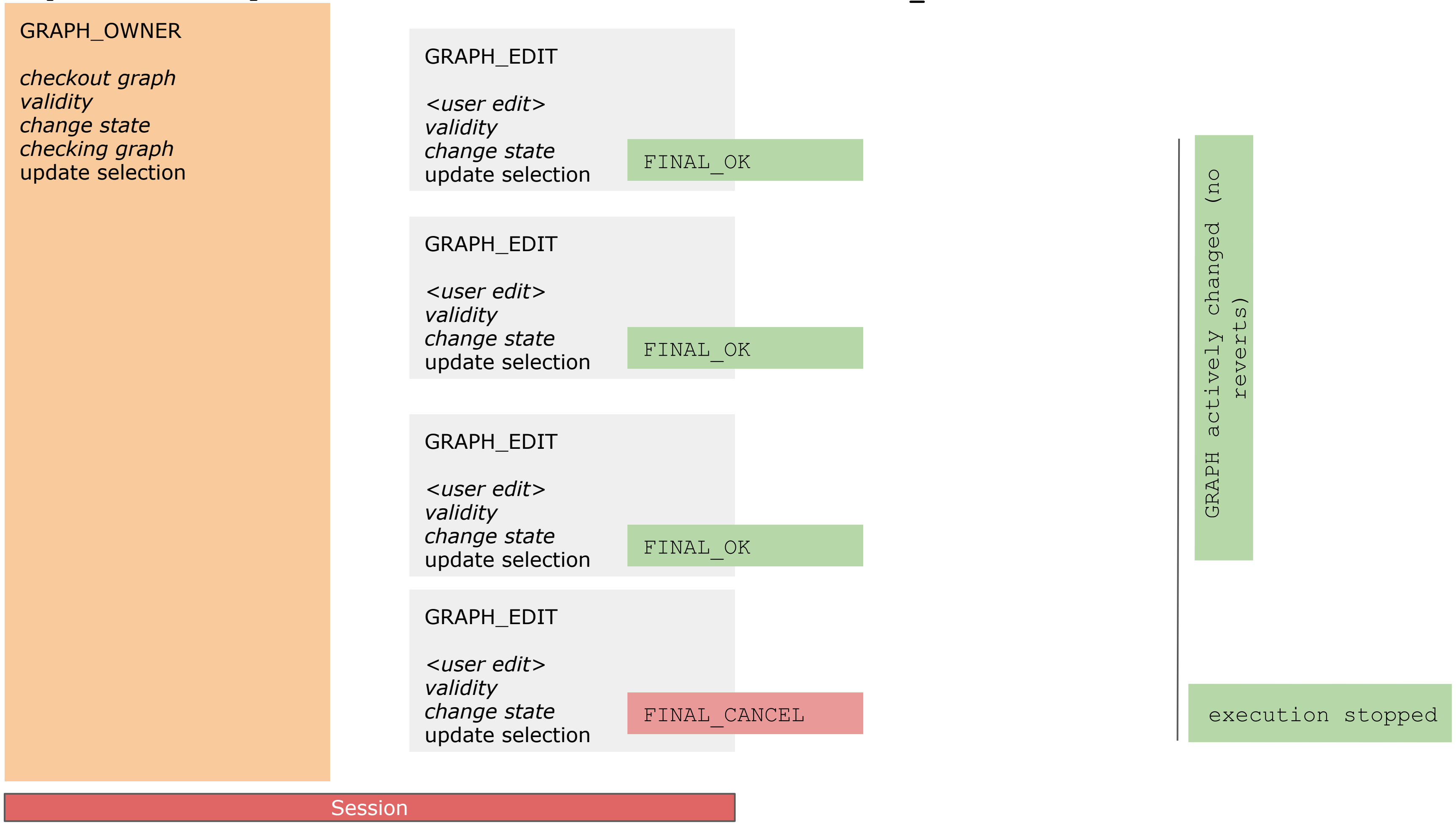


MEP2 - Multiple Execution Pattern (GRAPH_EDIT)

Execute a command multiple times on a*selected list of entities*
with stopping the execution due to cancel or ex

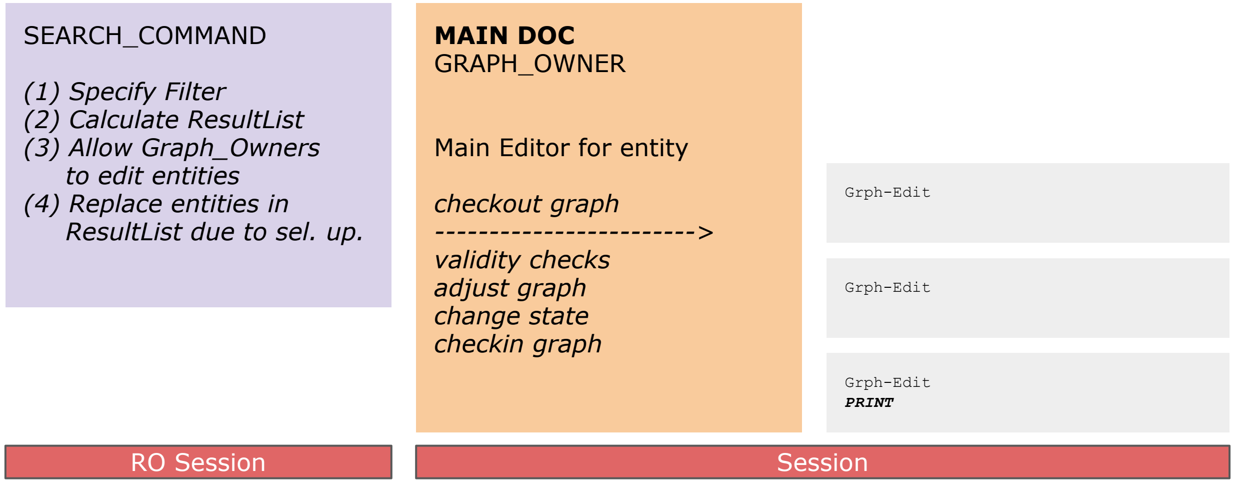
Command must not provide a session (GRAPH_EDIT)
Might have user interaction

Enabled condition for all Graph_Edits have to evaluate to true!
Keep care of hotkeys - F12, F12, F12, F12 - > save & close of GRAPH_OWNER !



PP - Print Pattern (aka Status Change Pattern)

Situation 1



SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

MAIN DOC
GRAPH_OWNER

Main Editor for entity

checkout graph
------------------------->
validity checks
adjust graph
change state
checkin graph

Grph-Edit

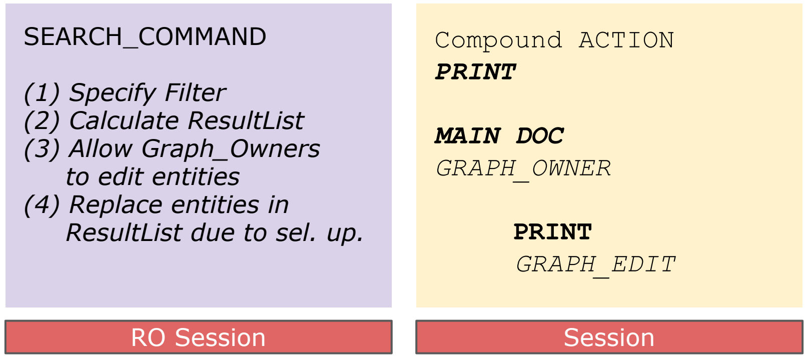
Grph-Edit

Grph-Edit
PRINT

RO Session

Session

Situation 2



SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

Compound ACTION
PRINT

MAIN DOC
GRAPH_OWNER

PRINT
GRAPH_EDIT

RO Session

Session

> **just a shortcut, solve once, use twice**
> no code duplication, no maintenance
> test once

> MAIN DOC has to contain all commands ever available anywhere.
> Compound action is canceled if GRAPH_EDIT does not eval to true!
> A “reason text” is calculated for the user message “Das Kommando kann im Zustand XXX nicht angewendet werden”
> only root selection is available, UI hierarchy not (but yes, right now it is…)

Background: PP II

Consistent Validation and Enabled/Disabled Evaluation of Commands

SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

RO Session

Compound ACTION
PRINT

MAIN DOC
GRAPH_OWNER

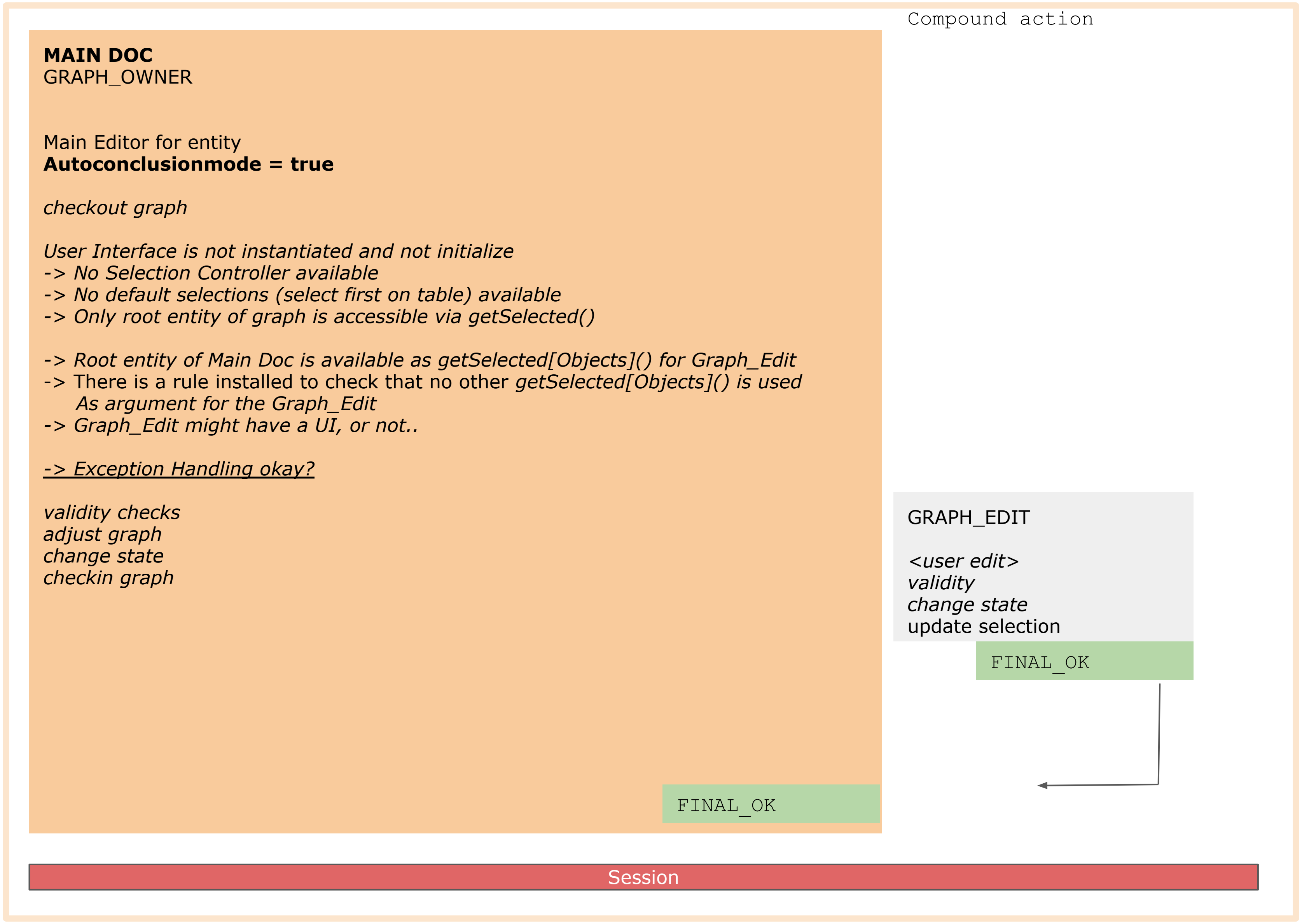
PRINT
GRAPH_EDIT

Session

```
> Statemachine does now try to indicate what the problem is,
  when disabling a command
> Permissions are also indicated in compound actions
> Enabled-If in commands is not indicate. Cancel would be more appropriate, e.g.
  Using preconditions to inform user: Cancel "You need admin permissions to .."

> Use same texts in tooltips in case a command is disabled
> Use same text to view messages in compound actions, when inner is disabled.
```

Background: Compound Action = No UI Handling

Situation 2



Compound action

MAIN DOC
GRAPH_OWNER

Main Editor for entity
Autoconclusionmode = true

checkout graph

User Interface is not instantiated and not initialize
-> No Selection Controller available
-> No default selections (select first on table) available
-> Only root entity of graph is accessible via getSelected()

-> Root entity of Main Doc is available as getSelected[Objects]() for Graph_Edit
-> There is a rule installed to check that no other *getSelected[Objects]() is used*
As argument for the Graph_Edit
-> Graph_Edit might have a UI, or not..

<u>-> Exception Handling okay?</u>

validity checks
adjust graph
change state
checkin graph

FINAL_OK

GRAPH_EDIT

<user edit>
validity
change state
update selection

FINAL_OK

Session

SGO - SubGraph Owner Pattern

Situation 1

SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

RO Session

SUB_GRAPH_OWNER
ADJUST SOMETHING
(with prompt window)

checkout/checkin

?LOCK? Session

> **might be a shortcut using a UI**
> no code duplication, no maintenance
> test once

> Any validation checks in GRAPH_OWNER with FLAG will be converted to CANCEL (= MsgBox)
> Is it necessary to view flags from GRAPH_OWNER in GRAPH_EDIT?

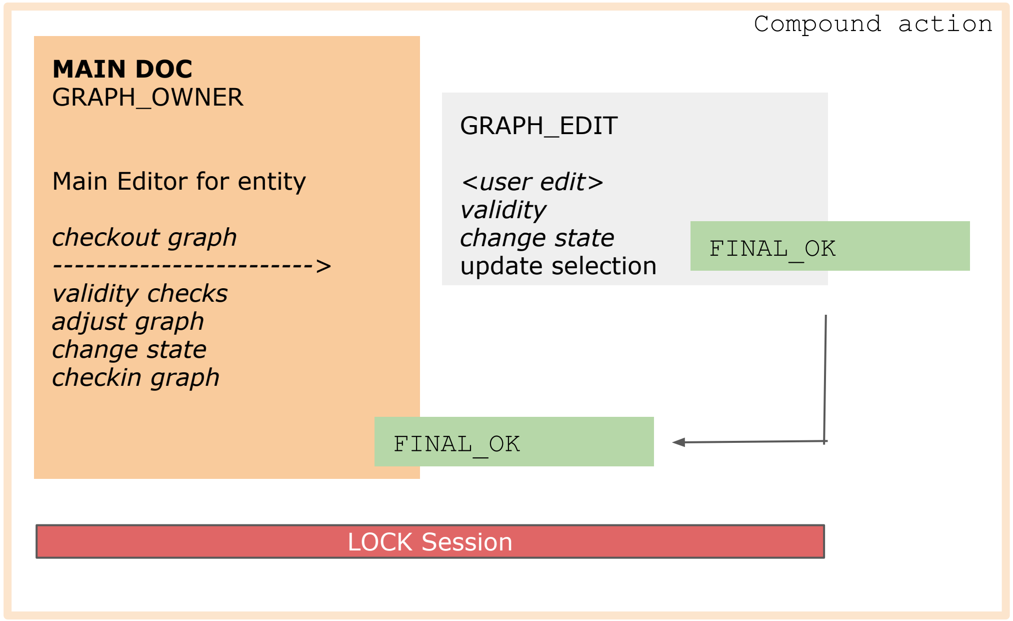
> Are the Unit of work boundaries ok? Anti-Pattern Search/Edit?

Situation 2

SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

RO Session



Compound action

MAIN DOC
GRAPH_OWNER

Main Editor for entity

checkout graph
------------------------->
validity checks
adjust graph
change state
checkin graph

FINAL_OK

GRAPH_EDIT

<user edit>
validity
change state
update selection

FINAL_OK

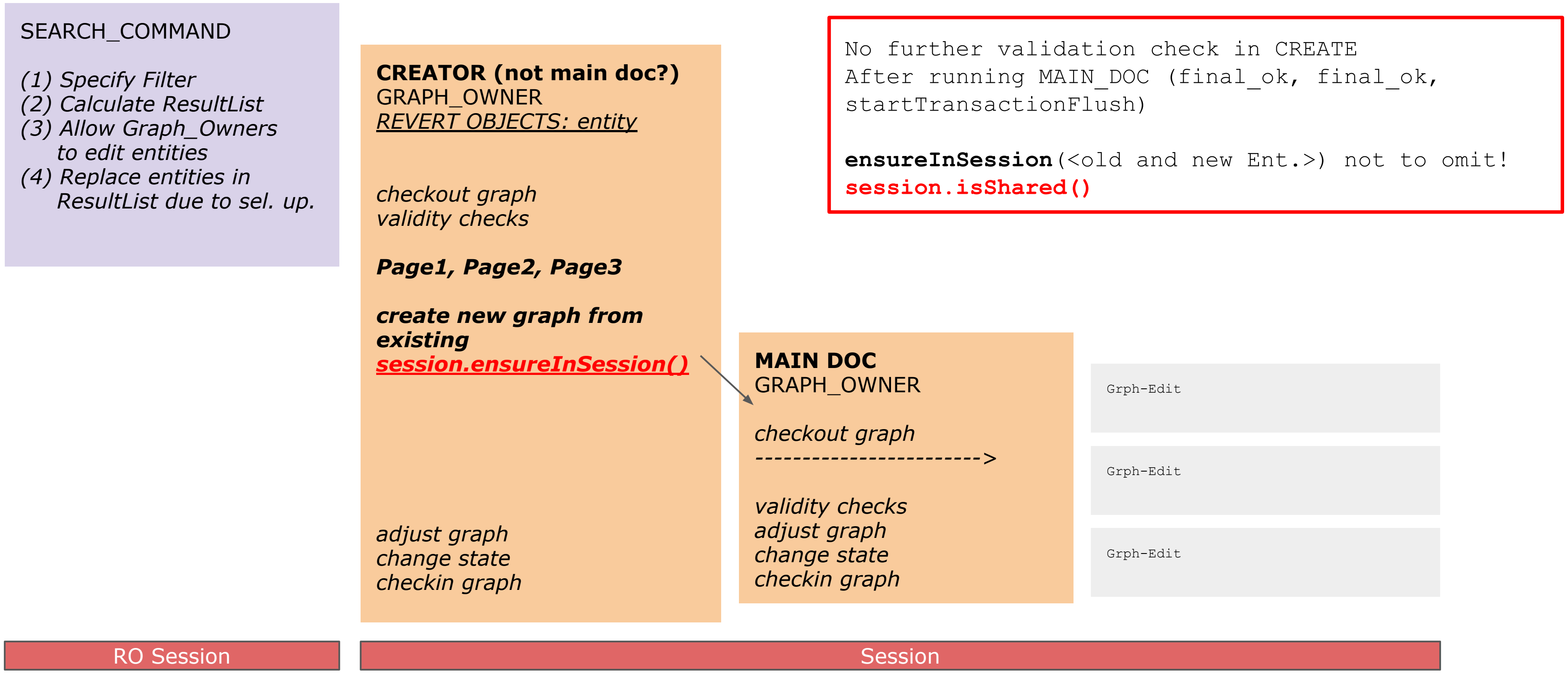
LOCK Session

CEP - Create Edit Pattern

(1) Checkout existing document
(2) Gather additional information (UI-Wizzard with multiple pages)
(3) Create a new complex graph

(4) Let user adjust more details by using the appropriate MAIN DOC editor
  -> Adjust & validate new graph
  -> ***<User can edit>***
  -> ***OK or CANCEL all - Revert***
  -> Adjust & validate graph
  -> FINAL_OK checkin new graph

(5) FINAL_OK (checkin existing document) OR FINAL_CANCEL (no EX)



SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

CREATOR (not main doc?)
GRAPH_OWNER
REVERT OBJECTS: entity

checkout graph
validity checks

Page1, Page2, Page3

create new graph from existing
session.ensureInSession()

adjust graph
change state
checkin graph

MAIN DOC
GRAPH_OWNER

checkout graph
------------------------>

validity checks
adjust graph
change state
checkin graph

No further validation check in CREATE
After running MAIN_DOC (final_ok, final_ok, startTransactionFlush)

ensureInSession(<old and new Ent.>) not to omit!
session.isShared()

MAIN_DOC Graph Owner

Edit Documents of a certain type - The**MAIN_DOC GRAPH_OWNER**

> Handles Session + ***checkout*** / checkin of graph
> Has a single Page
> Uses cancel, Flag and Conditions
> Must be available almost always! At least in RO mode!

Checkout and error-handling

MAIN DOC
GRAPH_OWNER

Main Editor for Entity-Graph

IF session.isShared()
sessionCheckedOut Enitity.size > 0

Entity.id == 0

> Do not checkout Graph again

IF not session.isShared()
sessionCheckedOut Enitity.size == 0

Entity.id > 0

> Checkout Graph for Main Doc
> Use Session-Debugger (CRTL-ALT F6) to see,
  if session is not marked as dirty

Preferred way o.D.

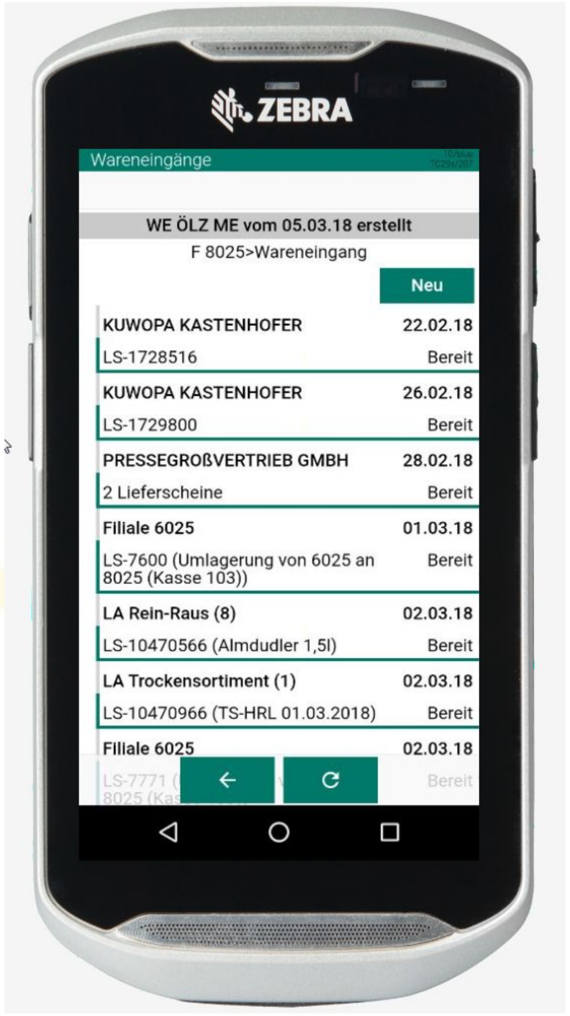
Allow for User edits

> Validate Graph in conclusion of “Save and Exit” by using flag
> Graph can also be validated by using flag in onChildTerm() to isolate all validation logic in a single point

> Use flattenGraph() / removeChildren() to clear lists<...> and references in head entity before
  applying pushSelection on parent

Command Creation Information

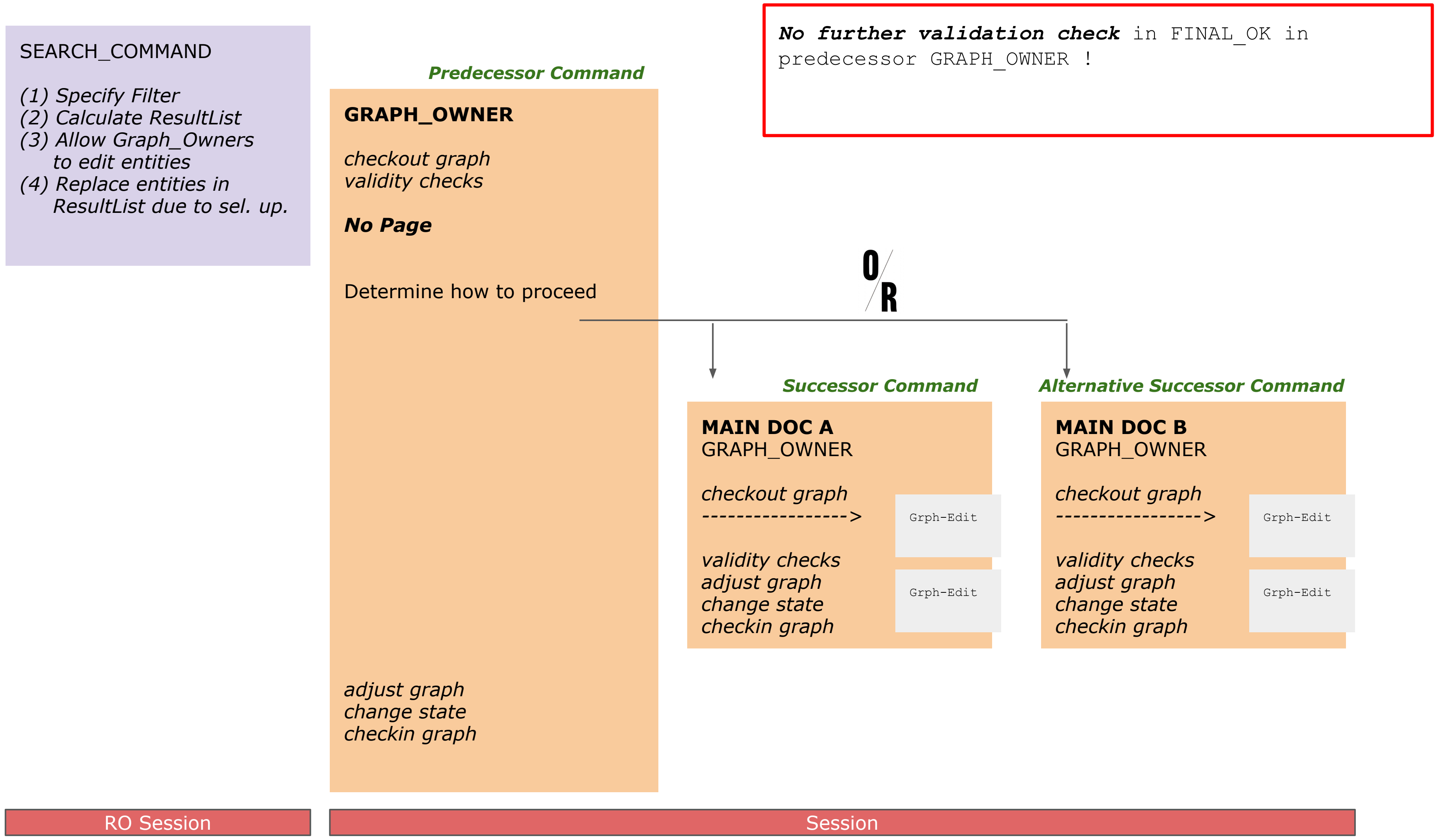
```
FINAL_OK_CONCLUSION:
  <sucsr cmds>

  func()->void {
    // check in order here!
    checkoutOrder.complete();
    // session.addOperation(new IOFXSessionOperation() {
      public void execute() {
        try {
          Thread.sleep(1000);
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
      public string getInformation() {
        "WAITING";
      }
    });
  }

  // check process, then commit session
  selection(s)/update(s) on parent: checkoutOrder.pos, checkoutOrder
  toast message 'Order %s (%d) erzeugt' % (checkoutOrder.name, checkoutOrder.id)
    (created/edited entity with key checkoutOrder.id as orderId)
```



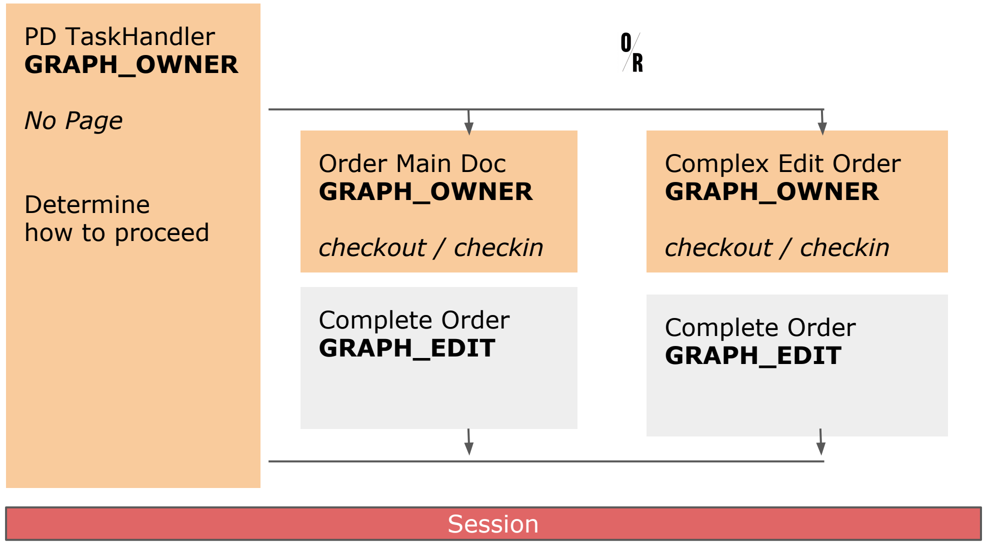
KUWOPA KASTENHOFER	22.02.18
LS-1728516	Bereit
KUWOPA KASTENHOFER	26.02.18
LS-1729800	Bereit
PRESSEGROßVERTRIEB GMBH	28.02.18
2 Lieferscheine	Bereit
Filiale 6025	01.03.18
LS-7600 (Umlagerung von 6025 an 8025 (Kasse 103))	Bereit
LA Rein-Raus (8)	02.03.18
LS-10470566 (Almdudler 1,5l)	Bereit
LA Trockensortiment (1)	02.03.18
LS-10470966 (TS-HRL 01.03.2018)	Bereit
Filiale 6025	02.03.18
LS-7771 (8025 (Kas	Bereit

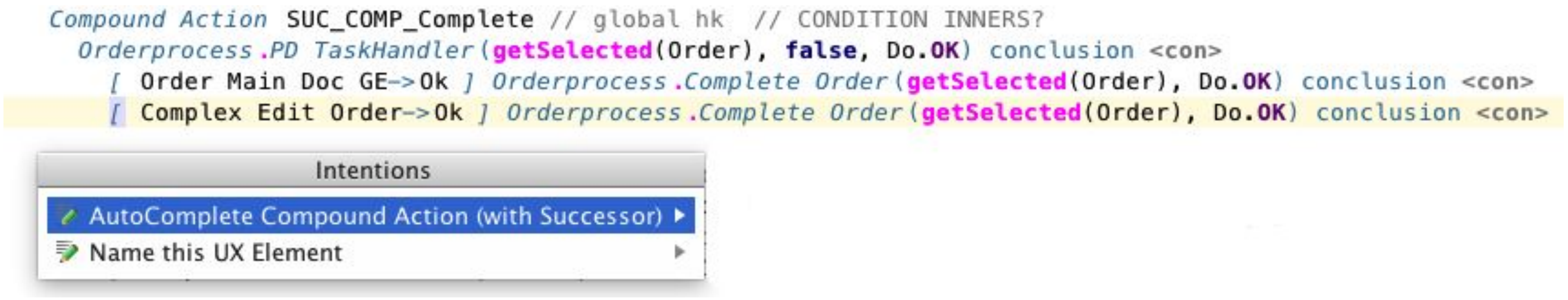
THP - Task Handling Pattern

```
(1)    Checkout entity (task) to determine how to proceed with different MAIN DOCS (documents)
(2)    Separate concerns between task and documents fully
(3)    Single session (unit of work) for both
(4)    BUT different procs and status checks for both
```



Compound action on successors





```
Compound Action SUC_COMP_Complete // global hk  // CONDITION INNERS?
  Orderprocess.PD TaskHandler(getSelected(Order), false, Do.OK) conclusion <con>
    [ Order Main Doc GE->Ok ] Orderprocess.Complete Order(getSelected(Order), Do.OK) conclusion <con>
    [ Complex Edit Order->Ok ] Orderprocess.Complete Order(getSelected(Order), Do.OK) conclusion <con>
```

A new programming model - sessionCheckedOut

CMD1: Task-Handler / Creator
CMD2: Main-Doc Editor

Problem: Passing information forward / backward from command to command
Solution: Using command arguments forward, passing info back with objects, references of those
passed forwards

Or even simpler: Working with shared SESSIONS

```
FINAL_OK {
       session.isShared()
       toSelect = sessionCheckedOut Aufgaben.last

}
selection(s) / updates(s) on parent: toSelect
```

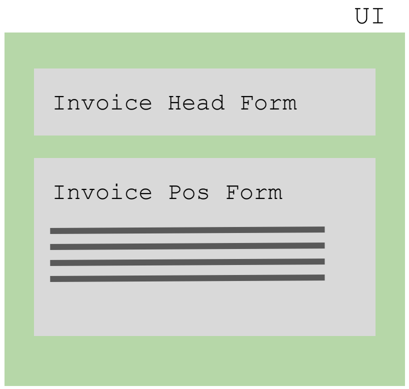
New: also entities saved are added to session as "checked out"

```
Existing Tasks: sessionCheckedOut Task
Created Task:session.ensureInSession(Entity / list<Entity>); // Existing or created
All Tasks in session: sessionCheckedOut Task
```

Handling Compositions

DOMAIN 1

> Edit single Invoice + Positions



MAIN DOC
GRAPH_OWNER

Main doc editor for "Invoice"

Grph-Edit

Edit Invoice Head

Grph-Edit

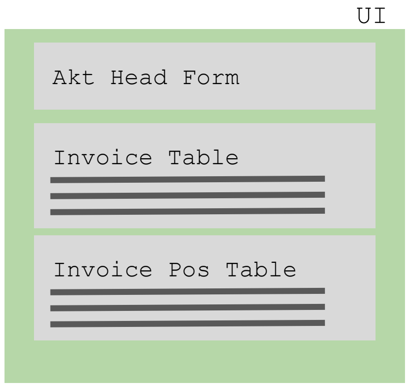
Edit Invoice Position

Complete INVOICE()

checkout Invoice
------------------------->
check validity
Complete graph info
checkin Invoice

DOMAIN 2

> Edit Invoice in RekoAkt Context
> Single Session



MAIN DOC
GRAPH_OWNER

Main doc editor for "RekoAkt"

Grph-Edit

Edit RekoAkt Head

validate()
Complete AKT ()

checkout Akt + Invoices
------------------------->
check validity
Complete graph info
checkin

Select Invoice in Table

Grph-Edit

Edit Invoice Head

Grph-Edit

Edit Invoice Position

GCP - Graph Composition Pattern

> Graph with compositions folder (Akt) contains multiple invoices (Rechnungen)
> Use the SAME graph_edit in multiple context, without violating "separation of concerns"
> Composition Pattern (changing an invoice in context folder, leads to changes in complete graph (folder) also
> Changing invoice in a context without a folder, can not lead to any changes

Situation 1

SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

MAIN DOC
GRAPH_OWNER

Main document editor for "Akt"

checkout Graph
------------------------>
check validity
complete graph info
checkin graph

Grph-Edit <u>on AKT</u>
- Akt validate
- Akt amend (calc diff, etc.)

Grph-Edit <u>on AKT</u>
(akt.complete() in final_ok)

Grph-Edit <u>on Invoice</u>

invoice.complete()

Grph-Edit <u>on Rechnung</u>
(Validation in GRAPH_EDIT possible,
Validation in Main Doc possible -> flag)

Situation 2

SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

MAIN DOC
GRAPH_OWNER

Main doc editor for "Invoice"

checkout Graph
------------------------>
check validity
Complete graph info
checkin graph

Grph-Edit <u>on Invoice</u>

invoice.complete()

Grph-Edit <u>on Invoice</u>

(Validation in GRAPH_EDIT possible,
Validation in Main Doc possible -> flag)

GCP - Graph Composition Pattern

Situation 1

SEARCH_COMMAND

(1) Specify Filter
(2) ResultList
(3) edit entities
(4) Replace entities

MAIN DOC
GRAPH_OWNER

Main document editor for “folder”

Page1 UI entry

***boundTo:* Graph**

-> scopes reevaluation

```
-> pageChildTerminatedFunc(termOk) {

        folder.complete()
        call BusinessLogic.Adjust(folder)

        FLAG
        CANCEL
}
```

checkout Graph
------------------------>
check validity
complete graph info
checkin graph

Grph-Edit on AKT
- Akt validate
- Akt amend (calc diff, etc.)

Grph-Edit **on AKT**
(akt.complete() in final_ok)

Grph-Edit on Invoice

invoice.complete()

Grph-Edit on Invoice
(Validation in GRAPH_EDIT possible,
 Validation in Main Doc possibl -> flag)

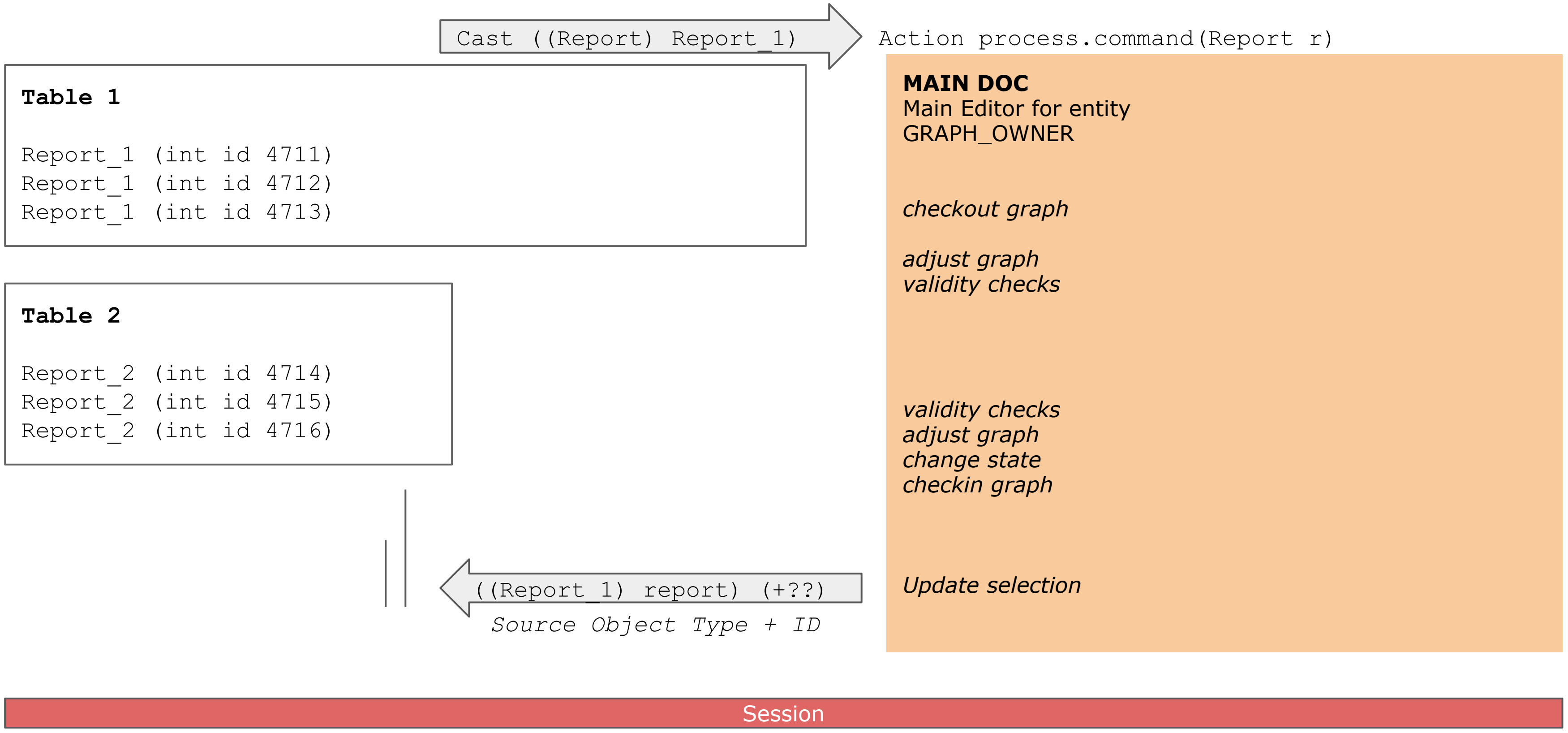
```
> Do not update or reload any lists, e.g. SEARCH_COMMAND :)
> flag and cancel are now available
```

BEP - Base Entity Pattern

Situation
```
Report1 extends Report
Report2 extends Report
```

Requirement
```
Use single command to edit Report1 & Report2
```

WHAT IF TWO Reports would be available?
Solution
```
getSelected(Report) +derived
getSelectedObjects(Report) +derived
```



```
Cast ((Report) Report_1)
```
```
Action process.command(Report r)
```

Table 1
```
Report_1 (int id 4711)
Report_1 (int id 4712)
Report_1 (int id 4713)
```

Table 2
```
Report_2 (int id 4714)
Report_2 (int id 4715)
Report_2 (int id 4716)
```

MAIN DOC
Main Editor for entity
GRAPH_OWNER

checkout graph

adjust graph
validity checks

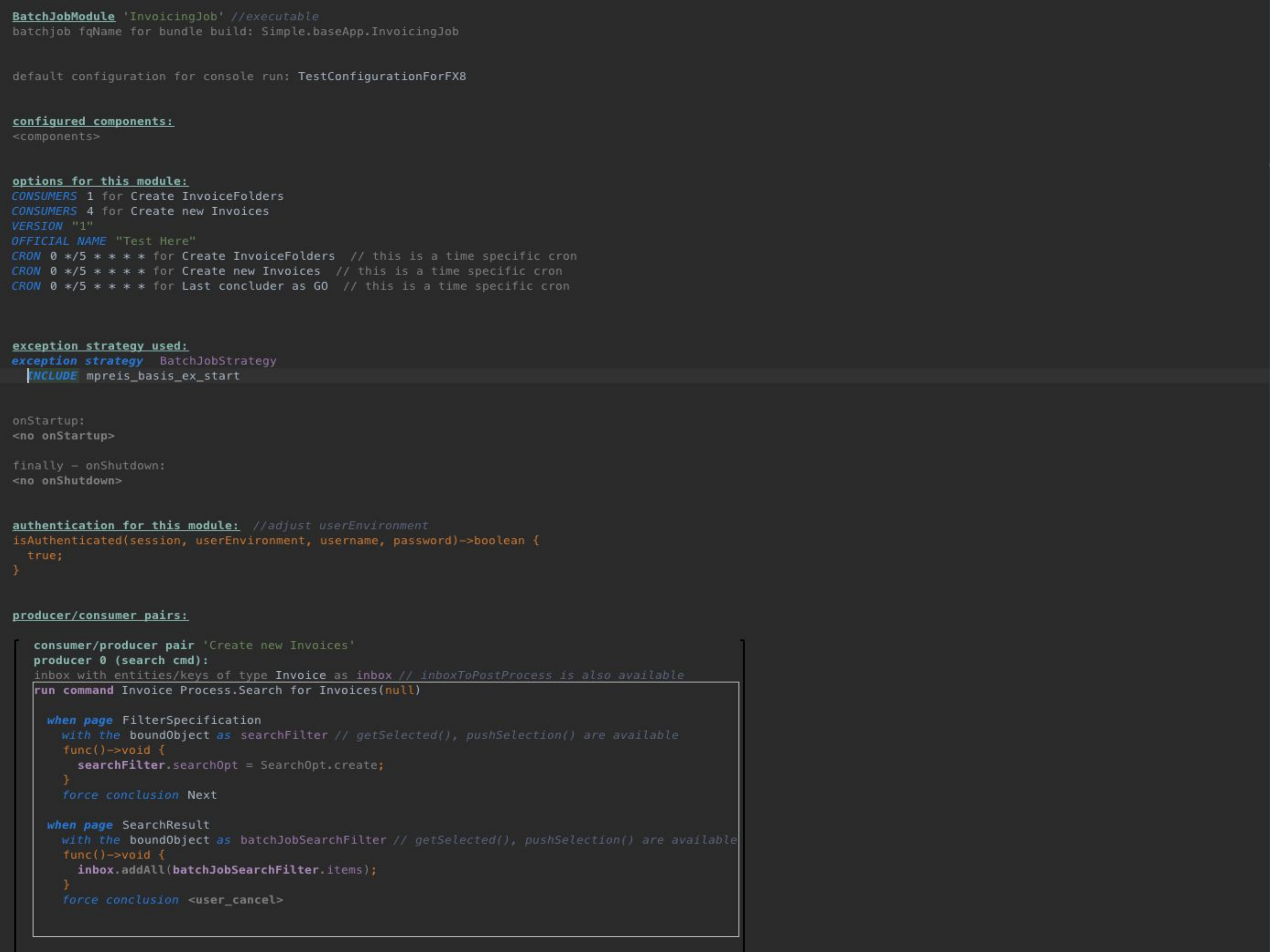
validity checks
adjust graph
change state
checkin graph

Update selection

```
((Report_1) report) (+??)
```
Source Object Type + ID

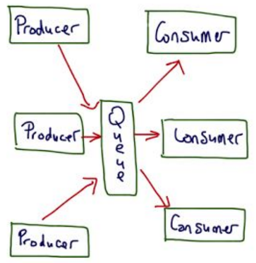
Session

MoWare CMD 2017



```
BatchJobModule 'InvoicingJob' //executable
batchjob fqName for bundle build: Simple.baseApp.InvoicingJob


default configuration for console run: TestConfigurationForFX8


configured components:
<components>


options for this module:
CONSUMERS 1 for Create InvoiceFolders
CONSUMERS 4 for Create new Invoices
VERSION "1"
OFFICIAL NAME "Test Here"
CRON 0 */5 * * * * for Create InvoiceFolders  // this is a time specific cron
CRON 0 */5 * * * * for Create new Invoices  // this is a time specific cron
CRON 0 */5 * * * * for Last concluder as GO  // this is a time specific cron



exception strategy used:
exception strategy  BatchJobStrategy
  INCLUDE mpreis_basis_ex_start


onStartup:
<no onStartup>

finally - onShutdown:
<no onShutdown>


authentication for this module:  //adjust userEnvironment
isAuthenticated(session, userEnvironment, username, password)->boolean {
  true;
}


producer/consumer pairs:

   consumer/producer pair 'Create new Invoices'
   producer 0 (search cmd):
   inbox with entities/keys of type Invoice as inbox // inboxToPostProcess is also available
   run command Invoice Process.Search for Invoices(null)

     when page FilterSpecification
       with the boundObject as searchFilter // getSelected(), pushSelection() are available
       func()->void {
         searchFilter.searchOpt = SearchOpt.create;
       }
       force conclusion Next

     when page SearchResult
       with the boundObject as batchJobSearchFilter // getSelected(), pushSelection() are available
       func()->void {
         inbox.addAll(batchJobSearchFilter.items);
       }
       force conclusion <user_cancel>
```

OFXBatchJob - WHY?



Common structure of all batchjobs

Write jobs like UI applications
with commands
-> producer is a SEARCH_COMMAND
(no transaction)
-> consumer is a GRAPH_OWNER with
commitable session

Develop, Experiment, Test and Document a
Batchjob UI first, experienceable

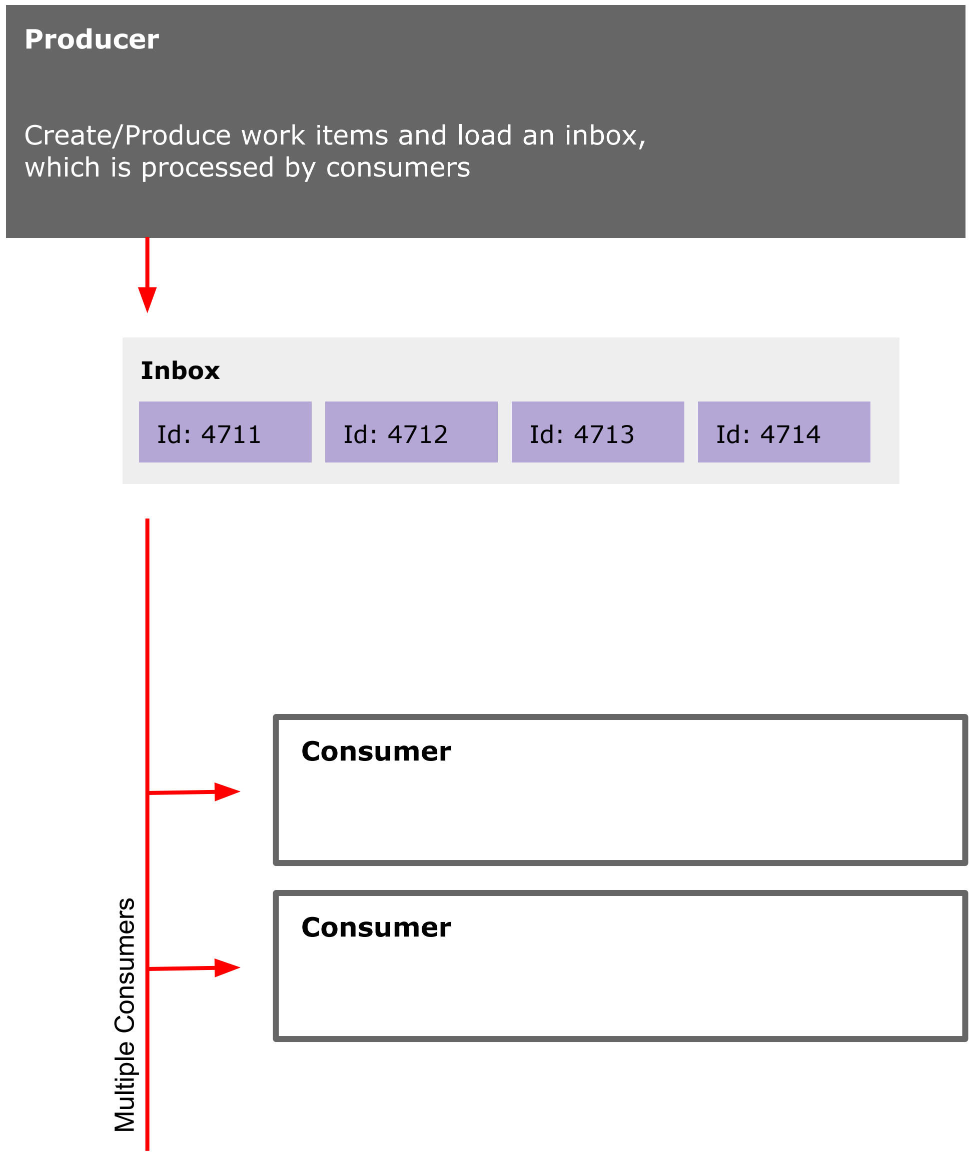
Structure job in small, comprehensible
Unit-Of-Work items

Specific exception handling on Unit-Of-Work
basis (never quit job)

Simple test the single Unit-Of-Work items
(straight from UI)

Use UI to manually work with job

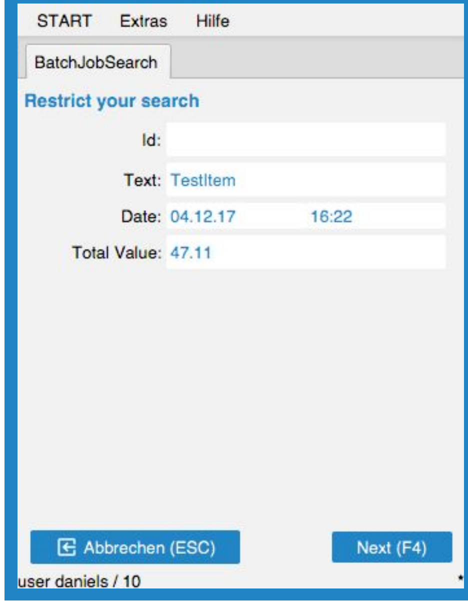
Allow for aktor like parallelization



OFXBatchJob - The UI FIRST approach

Producer

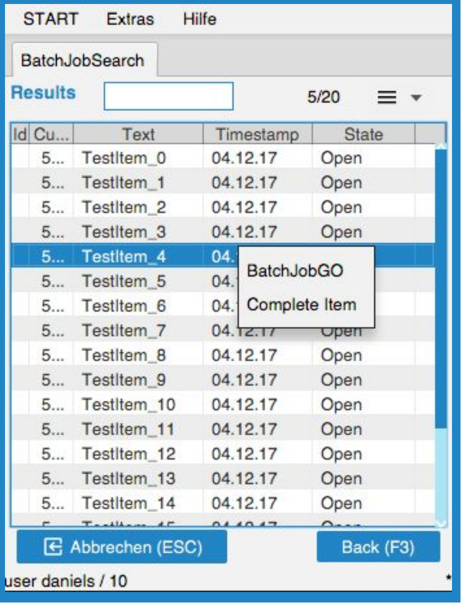
Search-Command (filter + result)



Restrict your search

Id:	
Text:	TestItem
Date:	04.12.17 16:22
Total Value:	47.11

user daniels / 10



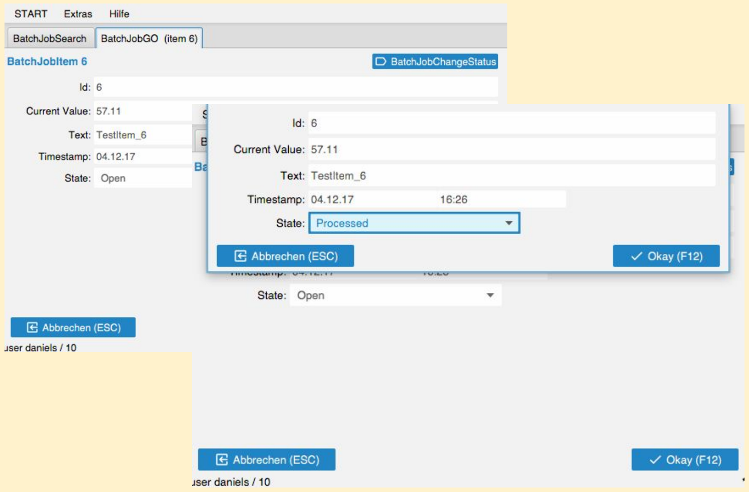
Results 5/20

Id	Cu...	Text	Timestamp	State
	5...	TestItem_0	04.12.17	Open
	5...	TestItem_1	04.12.17	Open
	5...	TestItem_2	04.12.17	Open
	5...	TestItem_3	04.12.17	Open
	5...	TestItem_4	04.	
	5...	TestItem_5	04.	
	5...	TestItem_6	04.	
	5...	TestItem_7	04.12.17	Open
	5...	TestItem_8	04.12.17	Open
	5...	TestItem_9	04.12.17	Open
	5...	TestItem_10	04.12.17	Open
	5...	TestItem_11	04.12.17	Open
	5...	TestItem_12	04.12.17	Open
	5...	TestItem_13	04.12.17	Open
	5...	TestItem_14	04.12.17	Open

user daniels / 10

Consumer

Graph-Owner + Successor



BatchJobItem 6

Id:	6
Current Value:	57.11
Text:	TestItem_6
Timestamp:	04.12.17
State:	Open

Id:	6
Current Value:	57.11
Text:	TestItem_6
Timestamp:	04.12.17 16:26
State:	Processed

State: Open

user daniels / 10

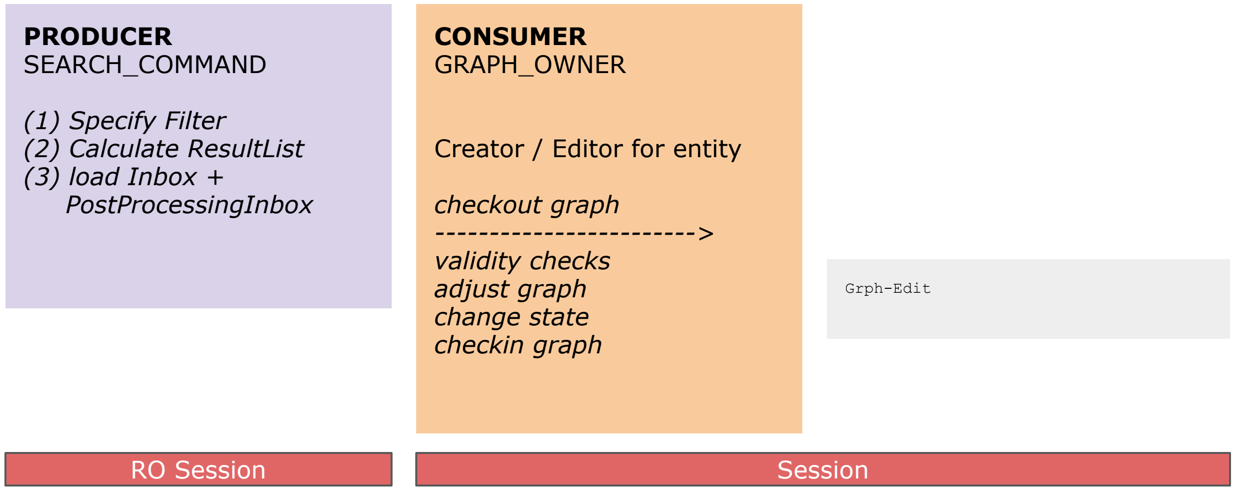
user daniels / 10

WorkItem +
Journal Entity

```
FINAL_OK:
commit graph
check process
update selection


FINAL_CANCEL (msg, ex):
commit cancel marker   (immediately)
commit journal marker  (immediately)

update selection
DO NOT check process !!
Log msg+ex in case of ex
```

OFXBatchJob - Session Handling

```
Producer = SEARCH_COMMAND (create/retrieve a list of Work items)
Consumer = GRAPH_OWNER + GRAPH_EDIT (with successor?)
```



PRODUCER
SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) load Inbox +
PostProcessingInbox

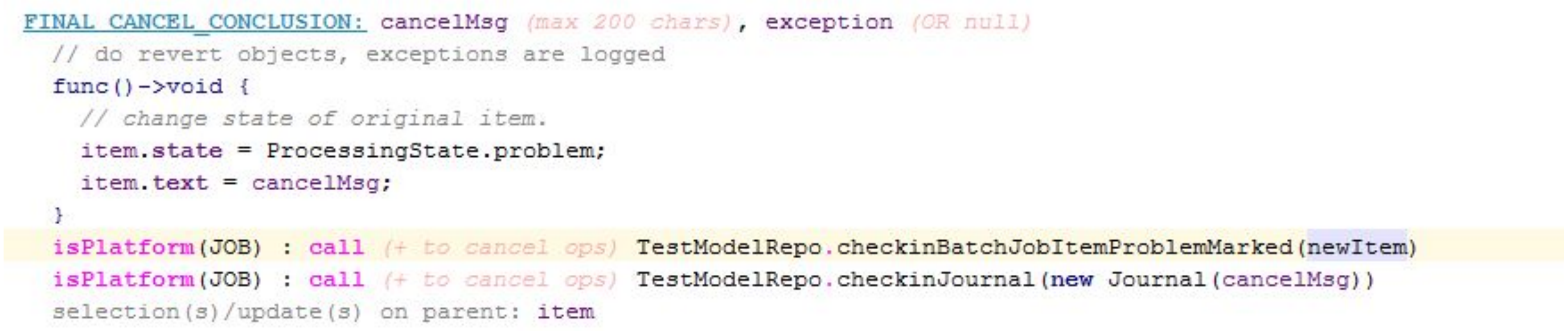
CONSUMER
GRAPH_OWNER

Creator / Editor for entity

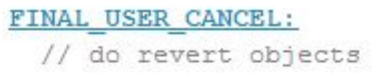
checkout graph
------------------------>
validity checks
adjust graph
change state
checkin graph

Ok	Graph_Owner + Graph_Edit -> final_ok
Cancel	Graph_Owner + Graph_Edit -> final_cancel
Error	Graph_Owner + Graph_Edit -> final_cancel
Tec. Exception	*Graph_Owner + Graph_Edit -> final_cancel + Log MSG*

OFXBatchJob - Graph_Owner Cancel Markers



```
FINAL_CANCEL_CONCLUSION: cancelMsg (max 200 chars), exception (OR null)
  // do revert objects, exceptions are logged
  func()->void {
    // change state of original item.
    item.state = ProcessingState.problem;
    item.text = cancelMsg;
  }
  isPlatform(JOB) : call (+ to cancel ops) TestModelRepo.checkinBatchJobItemProblemMarked(newItem)
  isPlatform(JOB) : call (+ to cancel ops) TestModelRepo.checkinJournal(new Journal(cancelMsg))
  selection(s)/update(s) on parent: item
```



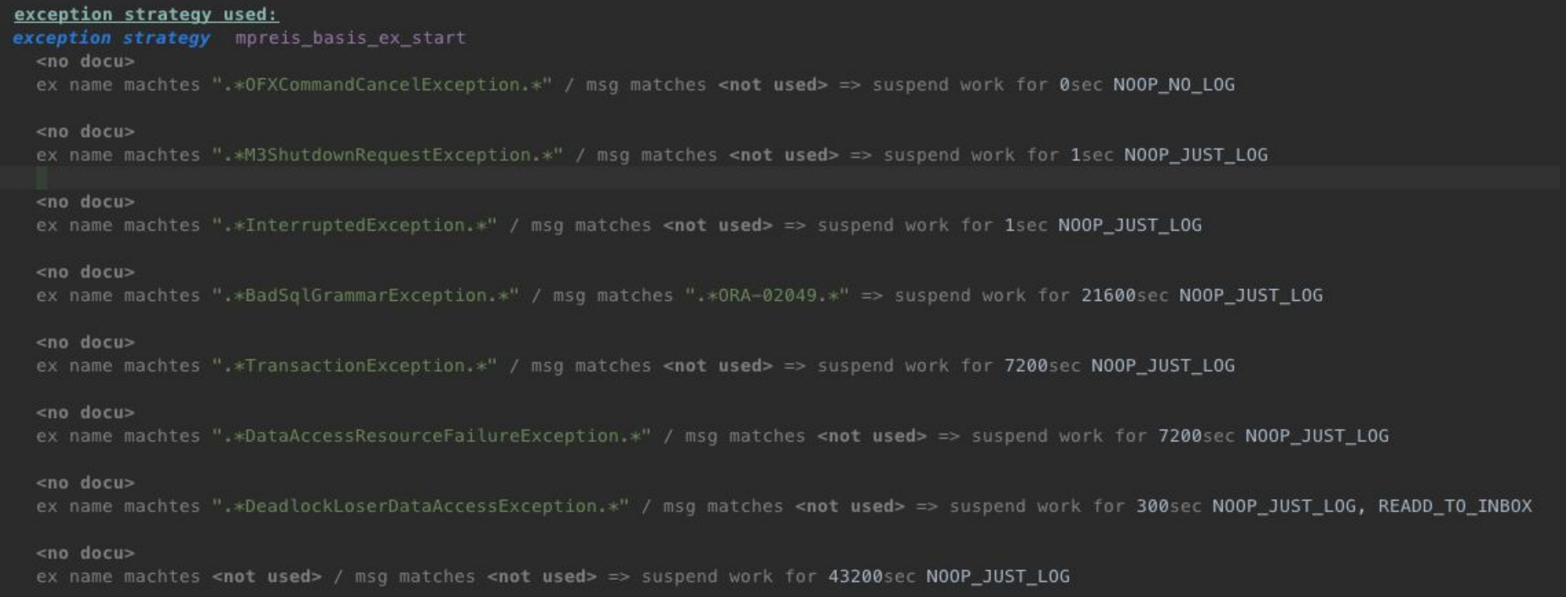
```
FINAL_USER_CANCEL:
  // do revert objects
```

Things to bear in mind:

- called in cases cancel statement is executed or exception occurs. (flag is translated to cancel if no ui present, like in jobs) Not called when user applies cancel button (-> final_user_cancel)
- In case of an exception, *ex parameter is not null* and the ex gets logged with log Error, including a stacktrace
- cancelMsg parameter contains cancel/flag statement text or ex name + msg in case of an exception
- cancelMsg is limmited to <200 chars to prevent any db field overflow when persisting
- Graph is reverted first, before final_cancel_conclusion method is executed
- Marker operations are immediately executed (together in one transaction) in case of an ex or a cancel (one can adopt the condition for specific behaviour, e.g. only on cancel)
- Clearly, markers will not work in case ex occurs due to db connection loss

OFXBatchJob - Exception Strategy

Idea:
- batchjobs should never crash by running into an exception.
- If an unknown problem occurs just pause for a defined time.
- Administrators can control behaviour via JMX (immediately run producer, Disable Producer etc.)



```
exception strategy used:
exception strategy  mpreis_basis_ex_start
  <no docu>
  ex name machtes ".*OFXCommandCancelException.*" / msg matches <not used> => suspend work for 0sec NOOP_NO_LOG

  <no docu>
  ex name machtes ".*M3ShutdownRequestException.*" / msg matches <not used> => suspend work for 1sec NOOP_JUST_LOG

  <no docu>
  ex name machtes ".*InterruptedException.*" / msg matches <not used> => suspend work for 1sec NOOP_JUST_LOG

  <no docu>
  ex name machtes ".*BadSqlGrammarException.*" / msg matches ".*ORA-02049.*" => suspend work for 21600sec NOOP_JUST_LOG

  <no docu>
  ex name machtes ".*TransactionException.*" / msg matches <not used> => suspend work for 7200sec NOOP_JUST_LOG

  <no docu>
  ex name machtes ".*DataAccessResourceFailureException.*" / msg matches <not used> => suspend work for 7200sec NOOP_JUST_LOG

  <no docu>
  ex name machtes ".*DeadlockLoserDataAccessException.*" / msg matches <not used> => suspend work for 300sec NOOP_JUST_LOG, READD_TO_INBOX

  <no docu>
  ex name machtes <not used> / msg matches <not used> => suspend work for 43200sec NOOP_JUST_LOG
```

> do not forget to include OFXCommandCancelException (technically not an ex but a cancel)
> match ex name (and optionally ex msg) with regular expressions
> specify “doNotWorkUntil” suspend time of job activity + action to take NOOP_JUST_LOG, NO_LOG, etc.
> component throwing ex will instantly stop (prod/cons), other consumers will commit UnitOfWork and stop

OFXBatchJobs - CronHandling

Continuous mode

> define one or multiple runtime periods, excluding service maintenance windows
> define an appropriate wait time, the delay-time

The job will be executed within the runtime periods. After completion of one cycle (producer + processing by consumers) the job will pause <delay-time> before running the producer again.

Time specific mode

> formulate one or multiple specific cron expressions, which will trigger at a specific point in time
> no delay-time

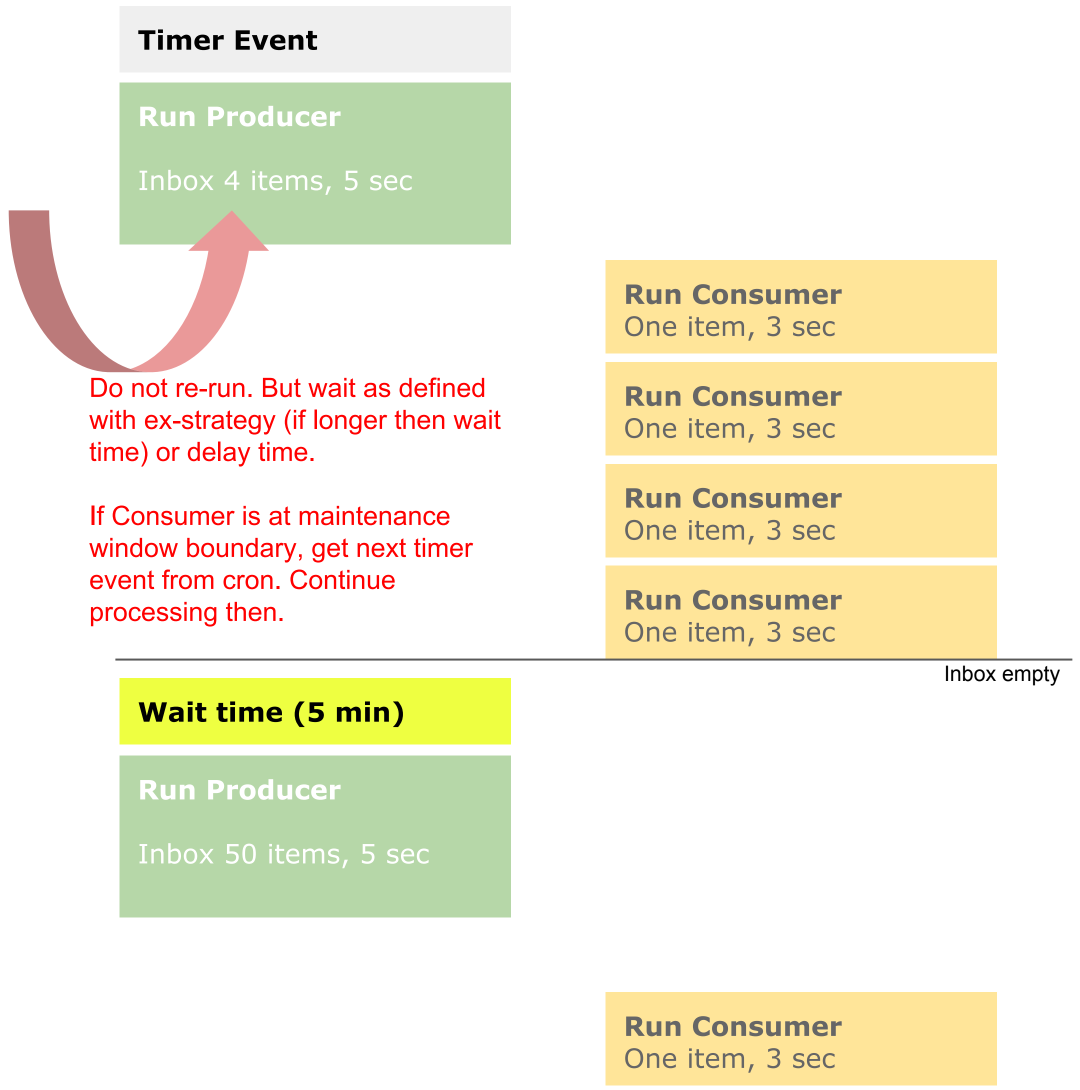
The job will be executed exactly at the defined cron times. If one cycle (producer + processing by consumers) ends, a new trigger time is drawn from cron.

If an exception occurs, a whole cycle (producer + processing by consumers) is not "restarted".
The job will pause according to the ex-strategy pause time, then wait for the next cron expression to trigger.

Cron Handling with Continuous Mode

```
cron setting: second 0 , minute */1 , hour 0-3,6-23 , day of month * , month * , day of week *
```

```
Run between 00:00 and
3:00 (am/pm)

Every minute, after
each full run.

  +   With a delay of
      5 Minutes when
      inbox is empty


* * 00-03 * * *
* * 12-15 * * *

Delay: 300
```

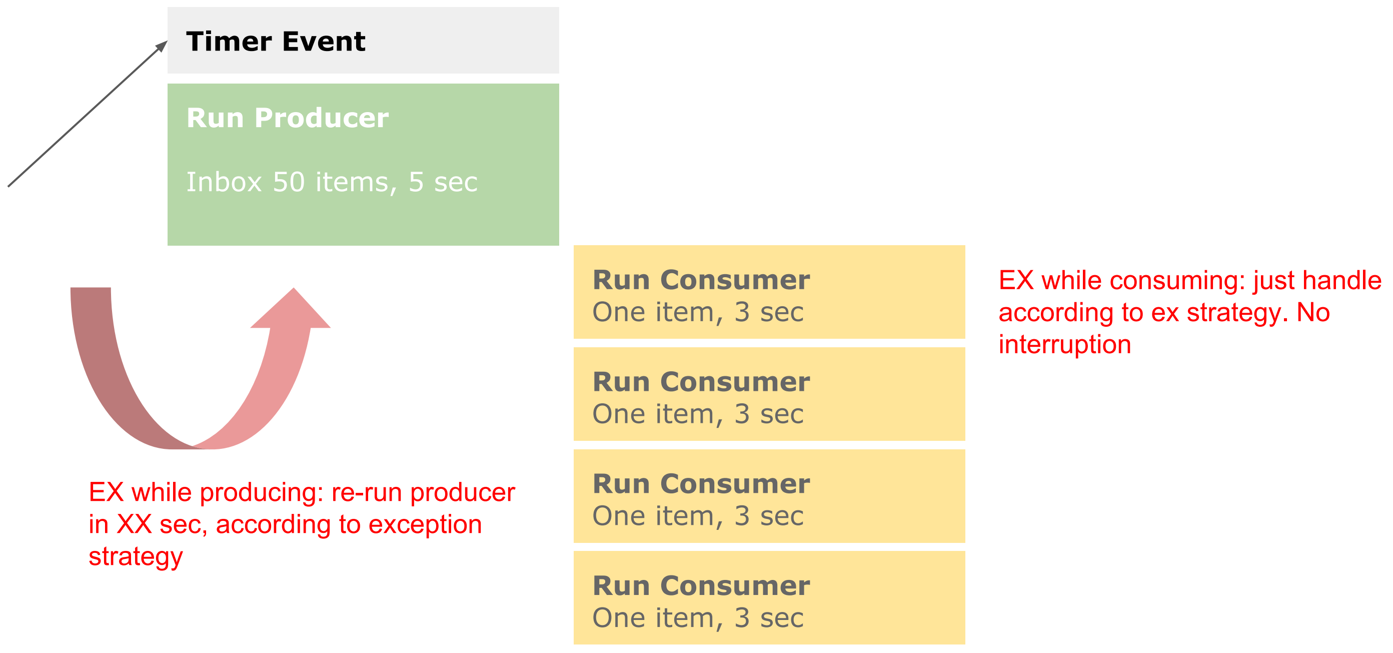


Do not re-run. But wait as defined with ex-strategy (if longer then wait time) or delay time.

If Consumer is at maintenance window boundary, get next timer event from cron. Continue processing then.

OFXBatchJob - CronHandling Mode I

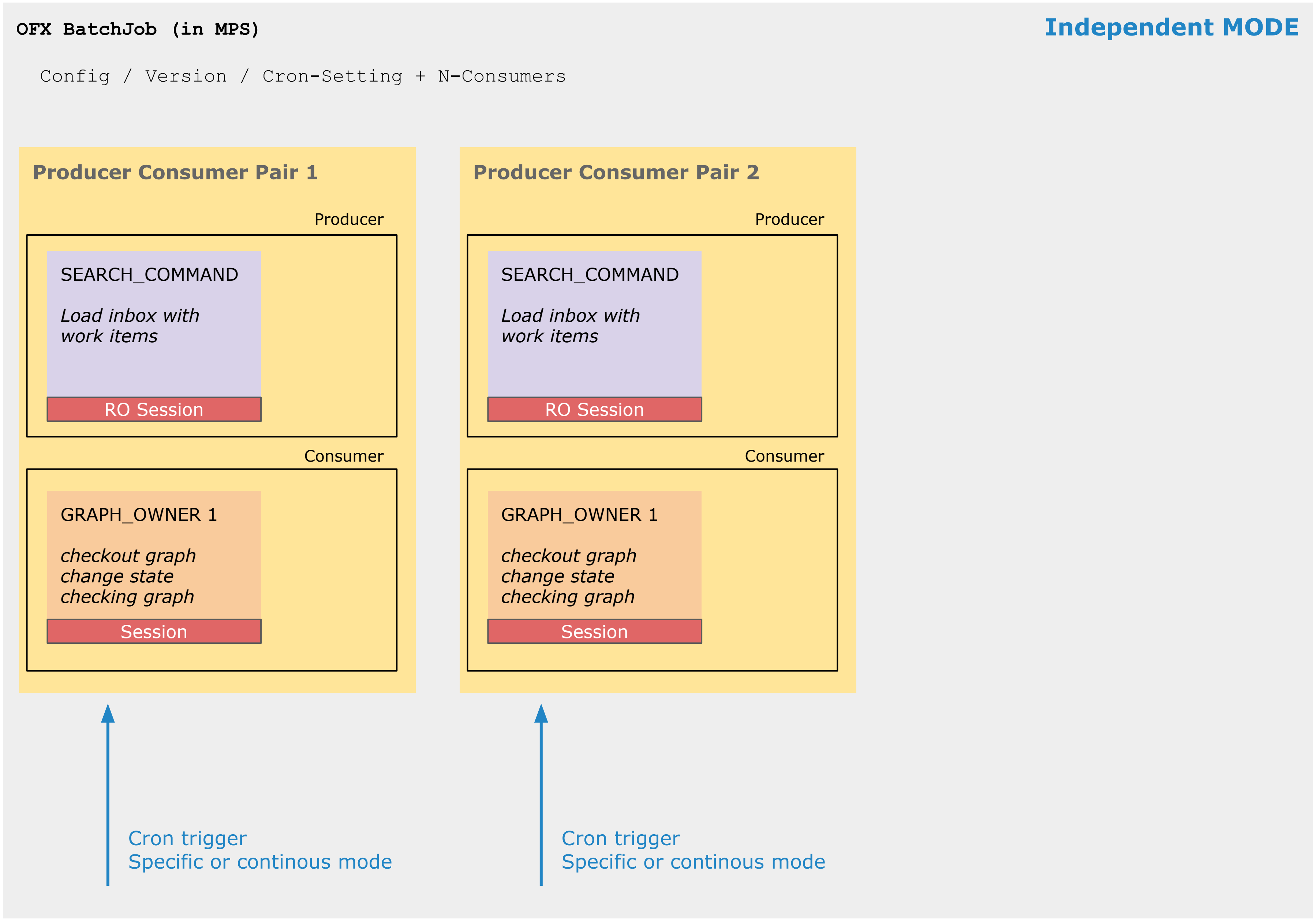
```
Run

00:00:00
12:00:00

0 0 0,12 * * *


(no dedicated service
time window, check
ex-strategies - it
never gives up… )
```



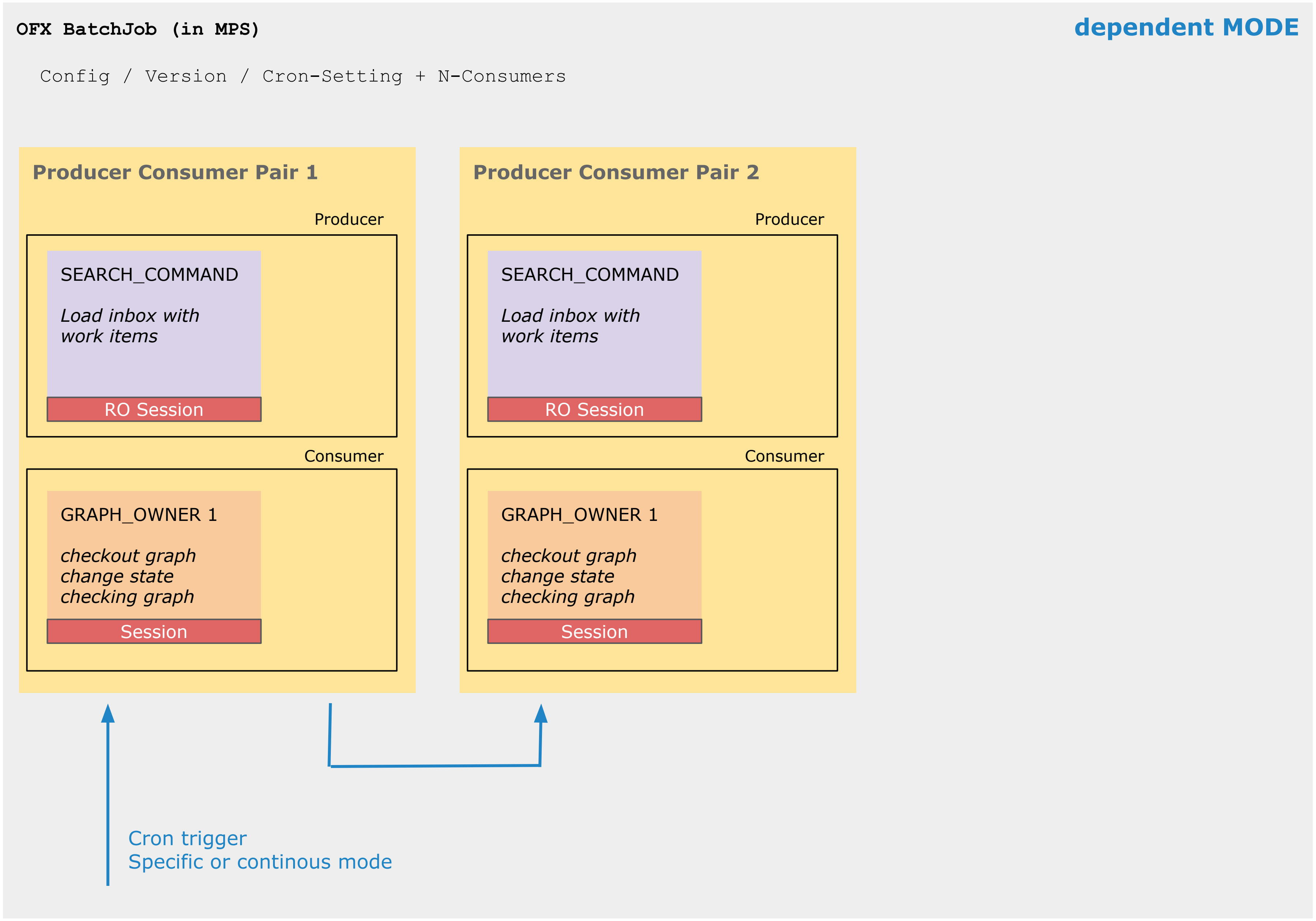
Run Producer

- No consumers should work (else error msg)
- Load inbox via search command
- Process all items with consumers
- Check next time to run producer again

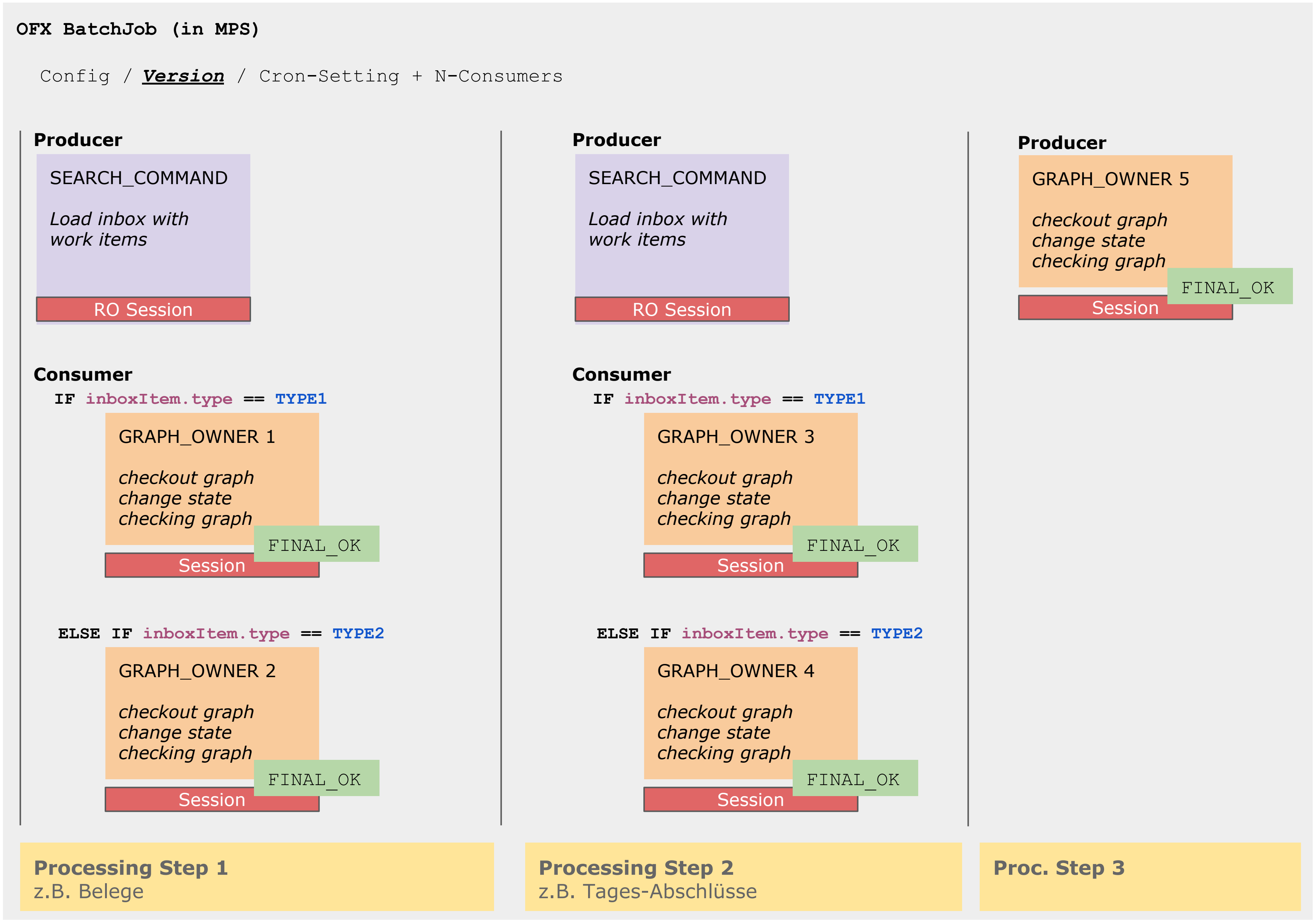
OFXBatchJob 2 Consumer Producer Pair



OFXBatchJob 2   Consumer Producer Pair



OFXBatchJob 2 - Proposition


OFX BatchJob (in MPS)
Config / ***Version*** / Cron-Setting + N-Consumers
Producer
SEARCH_COMMAND
Load inbox with work items
RO Session
Consumer
IF inboxItem.type == TYPE1
GRAPH_OWNER 1
checkout graph change state checking graph
FINAL_OK
Session
ELSE IF inboxItem.type == TYPE2
GRAPH_OWNER 2
checkout graph change state checking graph
FINAL_OK
Session
Processing Step 1
z.B. Belege
Producer
SEARCH_COMMAND
Load inbox with work items
RO Session
Consumer
IF inboxItem.type == TYPE1
GRAPH_OWNER 3
checkout graph change state checking graph
FINAL_OK
Session
ELSE IF inboxItem.type == TYPE2
GRAPH_OWNER 4
checkout graph change state checking graph
FINAL_OK
Session
Processing Step 2
z.B. Tages-Abschlüsse
Producer
GRAPH_OWNER 5
checkout graph change state checking graph
FINAL_OK
Session
Proc. Step 3

OFXBatchJob 2 - Pre & Post Processing (Use-Case)

Probably *only needed in case of any computational/practical restrictions* in order to keep unit-of-work small and manageable, e.g. large batch imports.
-> Keep Unit-Of-Work small and easily comprehensible
-> Otherwise, doesn't the single GO do the work?

OFX BatchJob (in MPS)

Config / Version / Cron-Setting + N-Consumers

Pre-Processing for "Task at hand" (Producer / Consumer pair)

Collect all bracketing documents and extract positions (sub-documents). Create leading (bracketing) document.

Processing for "Task at hand" (Producer / Consumer pair)

Process work items (might be created in pre-proc step) with multiple GOs without always querying similar to "if first item, create. etc."

Post-Processing for "Task at hand" (Producer / Consumer pair)

Summarize head documents, status changes, use aggregations on DB
E.g. summarize all "open" head documents.. etc…

Cron-Handling:
> Single Cron
> Run all pairs in a sequence
> Define n-consumers for each pair

Error-Handling:
> Only Consumer should run into EX, continue processing
> If EX in handling or while producing, wait and re-run producer!

OFXBatchJob 2 - Pre & Post Processing (Use-Case)

What can we assure with the new batch job handling, allowing consecutive producer/consumer pairs?

OFX BatchJob (in MPS)

```
Config / Version / Cron-Setting + N-Consumers
```

Producer/Consumer pair A

Collect all bracketing documents and extract positions (sub-documents). Create leading (bracketing) document.

Producer/Consumer pair B

Process work items (might be created in pre-proc step) with multiple GOs without always querying similar to "if first item, create. etc."

Producer/Consumer pair C

Summarize head documents, status changes, use aggregations on DB
E.g. summarize all "open" head documents.. etc…

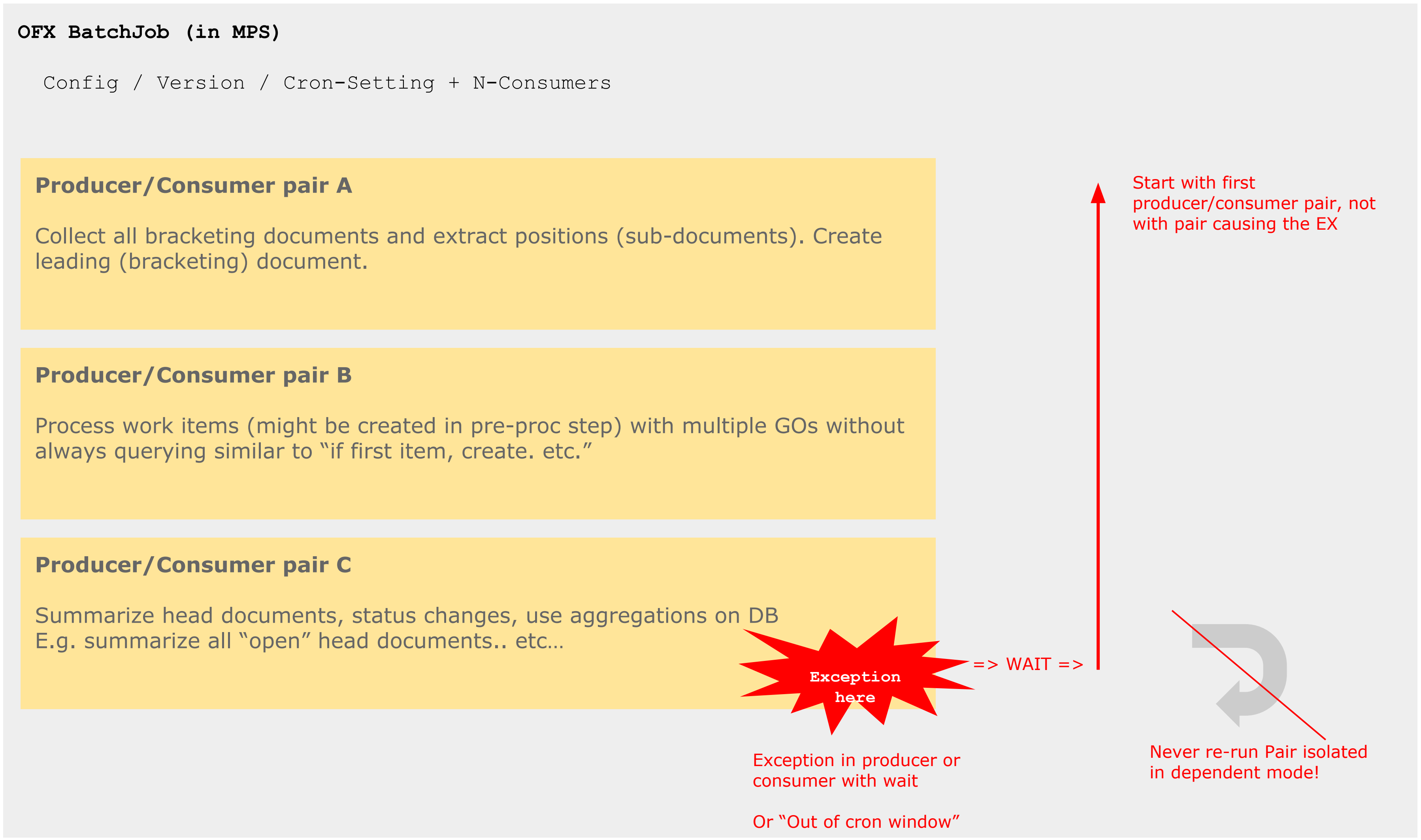
PC Pair B will never run without having PCPair A run successfully in advance

PC Pair C will never run without having PCPair B and A run successfully in advance

BUT: If we have Problems within A => B and C will not run at all !
THUS: Consumer EX -> Wait according to EX -> complete inbox then -> next producer/consumer pair

OFXBatchJob 2 - Pre & Post Processing Exception Handling

What can we assure with the new batch job handling, allowing consecutive producer/consumer pairs?



OFX BatchJob (in MPS)

```
  Config / Version / Cron-Setting + N-Consumers
```

Producer/Consumer pair A

Collect all bracketing documents and extract positions (sub-documents). Create leading (bracketing) document.

Producer/Consumer pair B

Process work items (might be created in pre-proc step) with multiple GOs without always querying similar to “if first item, create. etc.”

Producer/Consumer pair C

Summarize head documents, status changes, use aggregations on DB
E.g. summarize all “open” head documents.. etc…

Start with first producer/consumer pair, not with pair causing the EX

Exception in producer or consumer with wait

Or “Out of cron window”

Never re-run Pair isolated in dependent mode!

OFXConsumerProducerPair

EX Settings now?

Job Controller

- Cron Handling
- Throttle mode handling
- Run next consumer/producer after consumer producer done.
- Run support for Job
-

Producer/Consumer Pair

JMX Reporting

> Manual Run Producer
> Disable Producer
> Logs / Traces

Producer/Consumer Pair

Producer / Search command
Consumer / GO Command

EX in Producer? -> Inbox empty
- Exit Job?
- Apply waiting time?
- Resched new run for JOB !

EX in Consumer?
- Exit Job?
- Proceed/Ignore (re-add inbox)
- Apply waiting time?
- Resched continuation? OR
- Resched new run for JOB !

JMX Reporting

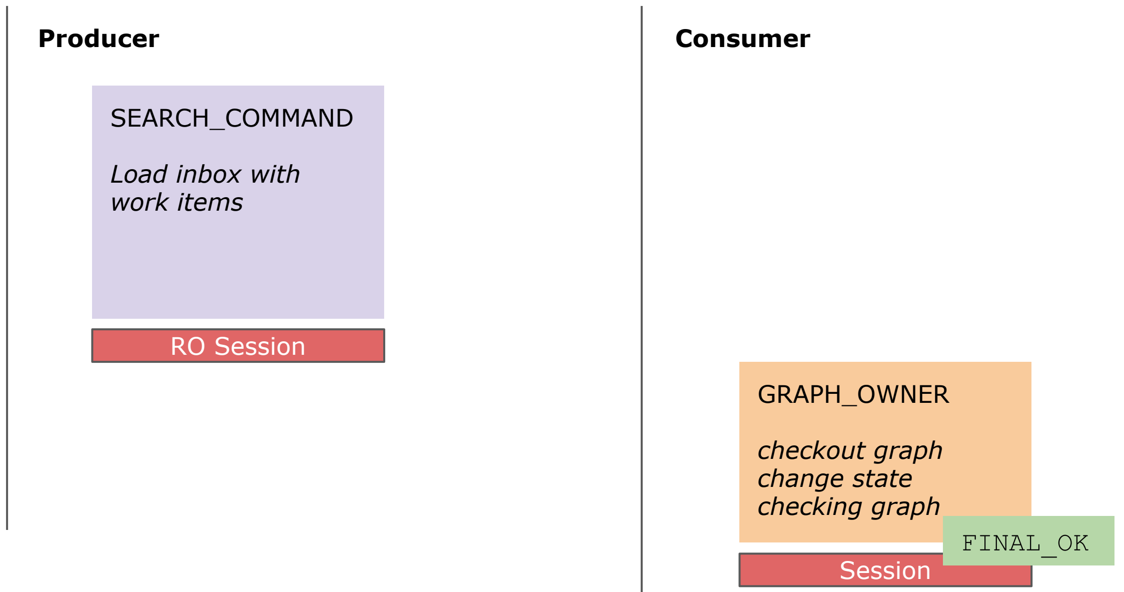
> Manual Run Producer
> Disable Producer
> Logs / Traces

Producer/Consumer Pair

JMX Reporting

> Manual Run Producer
> Disable Producer
> Logs / Traces

OFXBatchJob - Producer/Consumer Pair



- SEARCH_COMMAND calculates result-list which is loaded as inbox (items can be loaded as entities from DB or items can be created on the fly without persisting them - list of ViewObjects)
- single GRAPH_OWNER is consumer to process inbox by using on session per inbox-item and unit-of work
- Successor-Pattern is often seen in practice

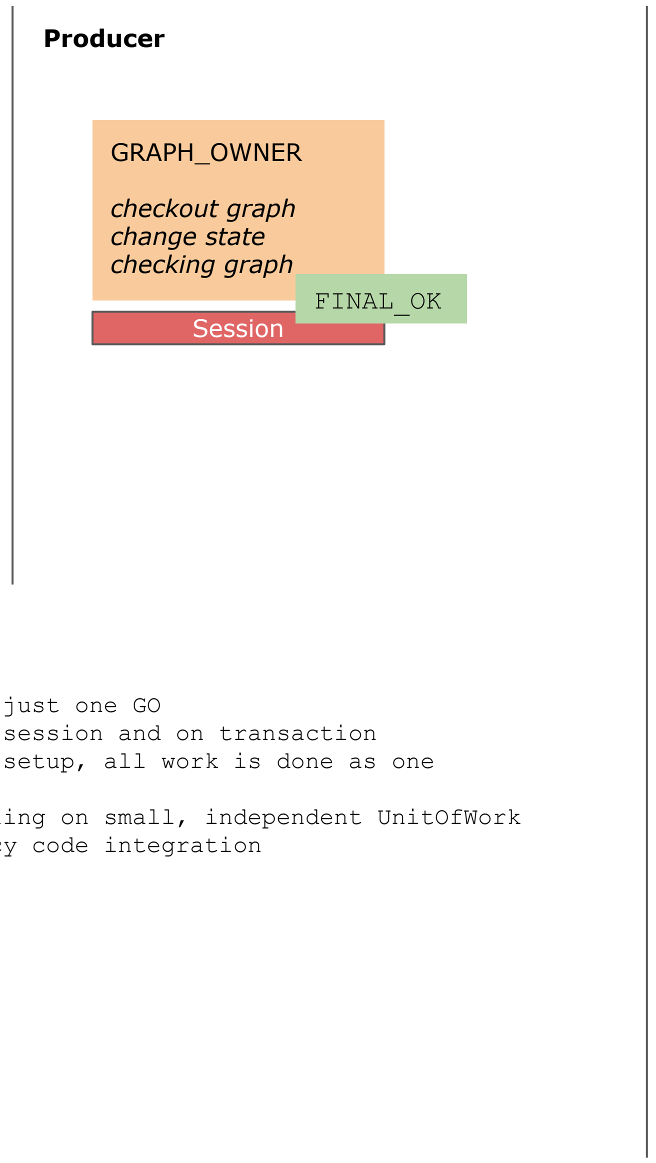
  ```
  create document BBB from document AAA
  ```

- Only via GRAPH_OWNER transaction are work-items marked as done or as "in error state" (*final_ok_conclusion* transaction or *final_cancel_conclusion* markers)

=> multiple consumers can be instantiated during runtime to process inbox-items in parallel with the same GRAPH_OWNER
=> well defined ex behaviour by ex strategy

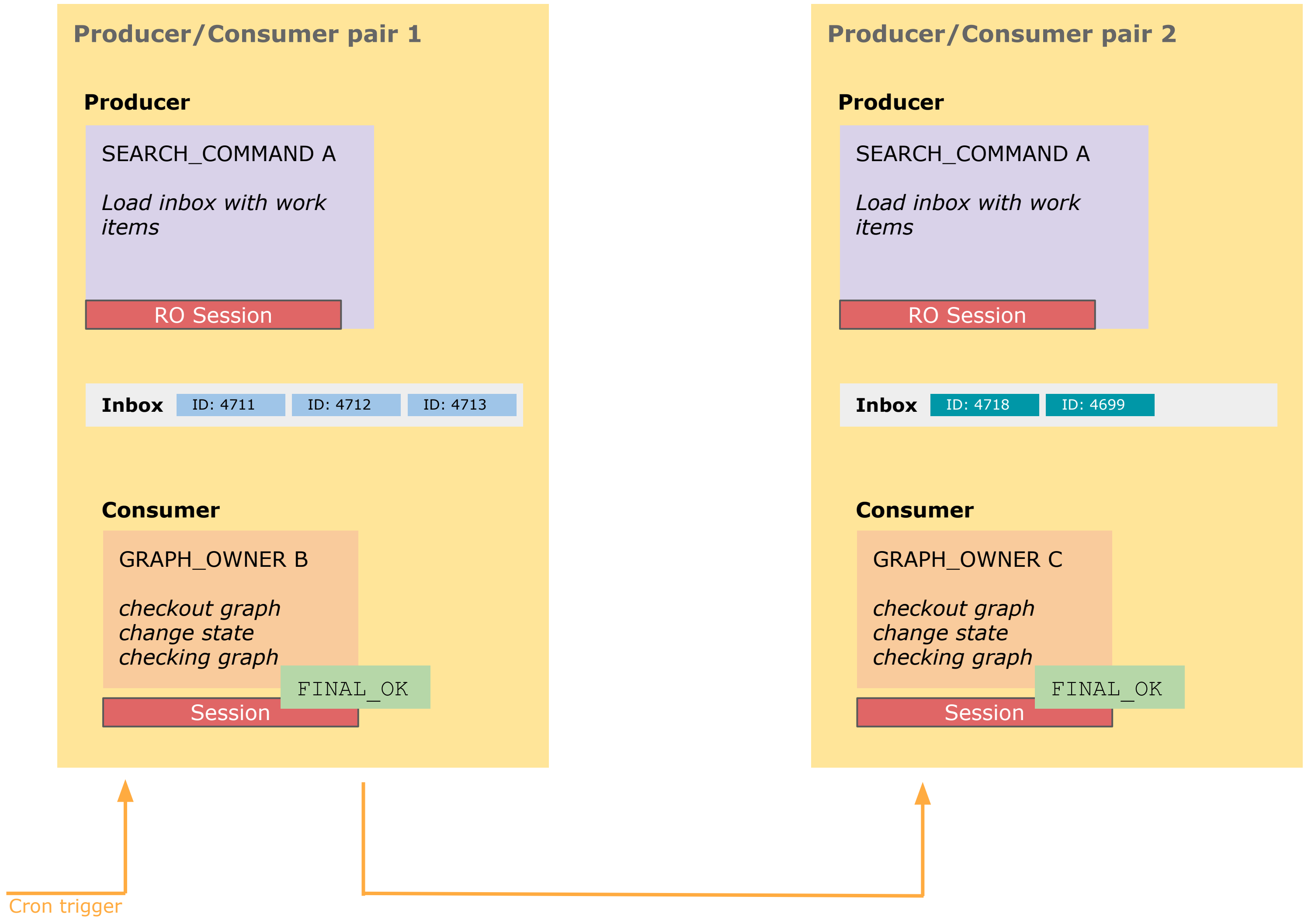
! Apply cancel in command_init of GRAPH_OWNER to check, if work-item was already processed in the meantime! (e.g. by a user manually via UI)

OFXBatchJob - Producer / Consumer Pair without Consumer



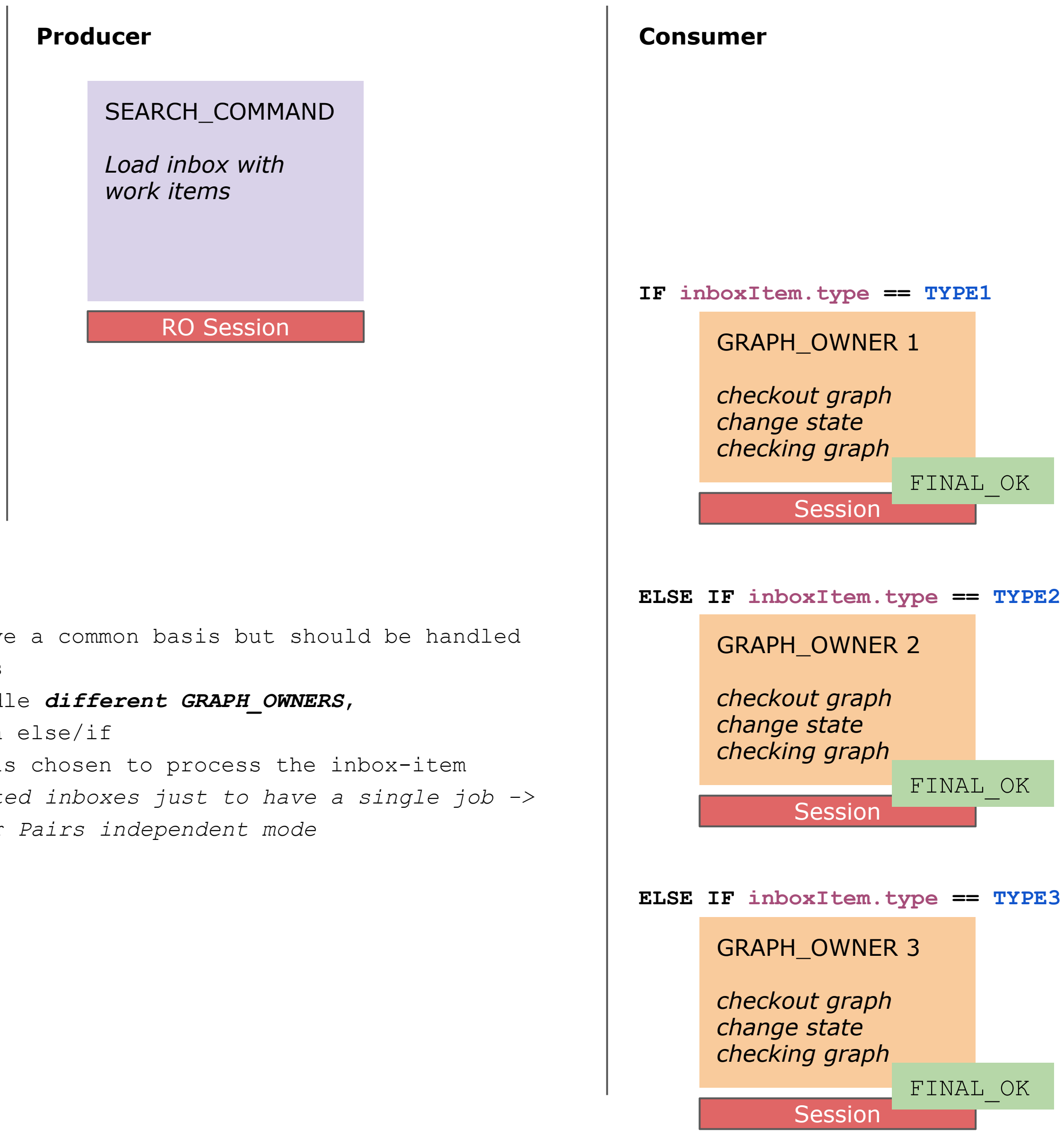
- No Producer / Consumer setup, just one GO
- Single GO comes with a single session and on transaction
- GO is executed according cron setup, all work is done as one UnitOfWork
- No parallelization or ex handling on small, independent UnitOfWork
- Probably only useful for legacy code integration

OFXBatchJob - Dependent Producer/Consumer Pair



- Ensure that Items of Type B are processed after Items of Type A
- Due to parallelization constraints two inboxes are needed
- -> Use two Producer/Consumer Pairs
- Former solved by “InboxToPostProcess”

OFXBatchJob - Producer and Consumer choosing GO



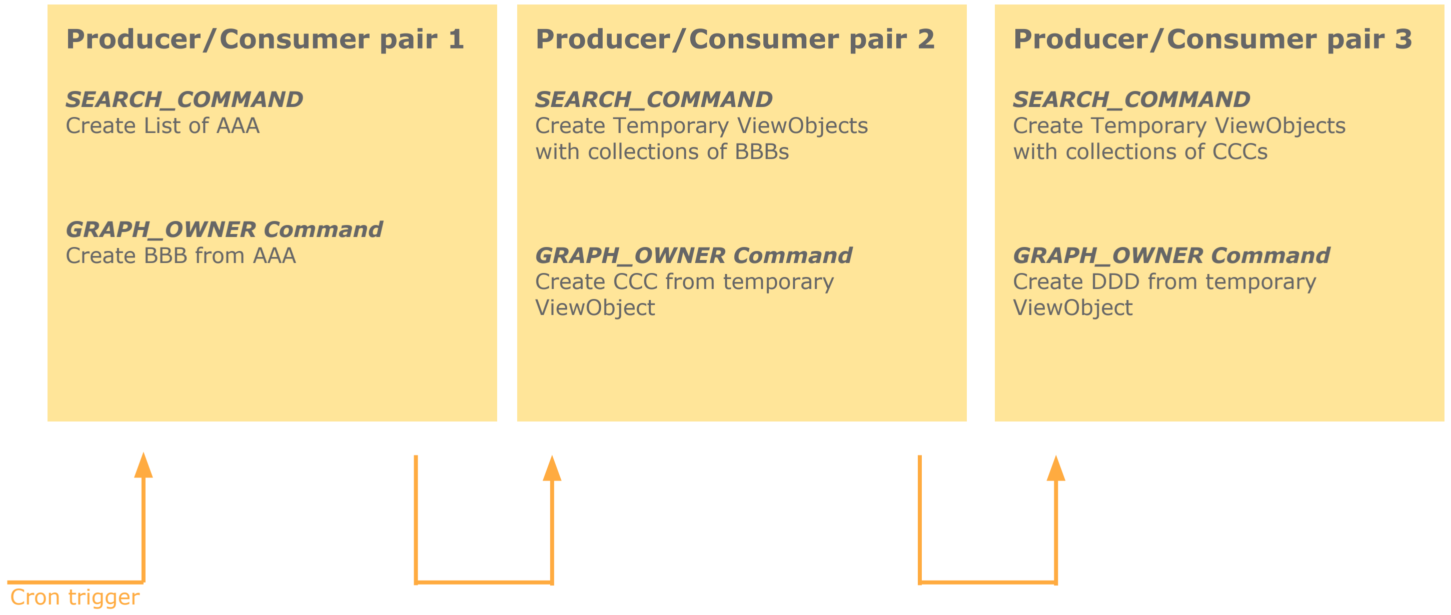
- Inbox contains items which have a common basis but should be handled differently according a status
- One consumer instance can handle ***different GRAPH_OWNERS***, calling one of them based on a else/if
- Only one of the GRAPH_OWNERS is chosen to process the inbox-item
- *However: Do not build aggregated inboxes just to have a single job -> use multiple Producer/Consumer Pairs independent mode*

OFXBatchJob - Multiple Producer/Consumer Pairs “Chain of Documents”

- Create dependent Artifacts in a consecutive manner
- Create Artefacts DDD from CCCs, which are in turn generated from BBBs

```
create document BBB from document AAA
create document CCC from multiple BBB documents
create document DDD from multiple CCC document
```

- *Idea: Use multiple dependent Producer/Consumer Pairs*



Producer/Consumer pair 1

SEARCH_COMMAND
Create List of AAA

GRAPH_OWNER Command
Create BBB from AAA

Producer/Consumer pair 2

SEARCH_COMMAND
Create Temporary ViewObjects with collections of BBBs

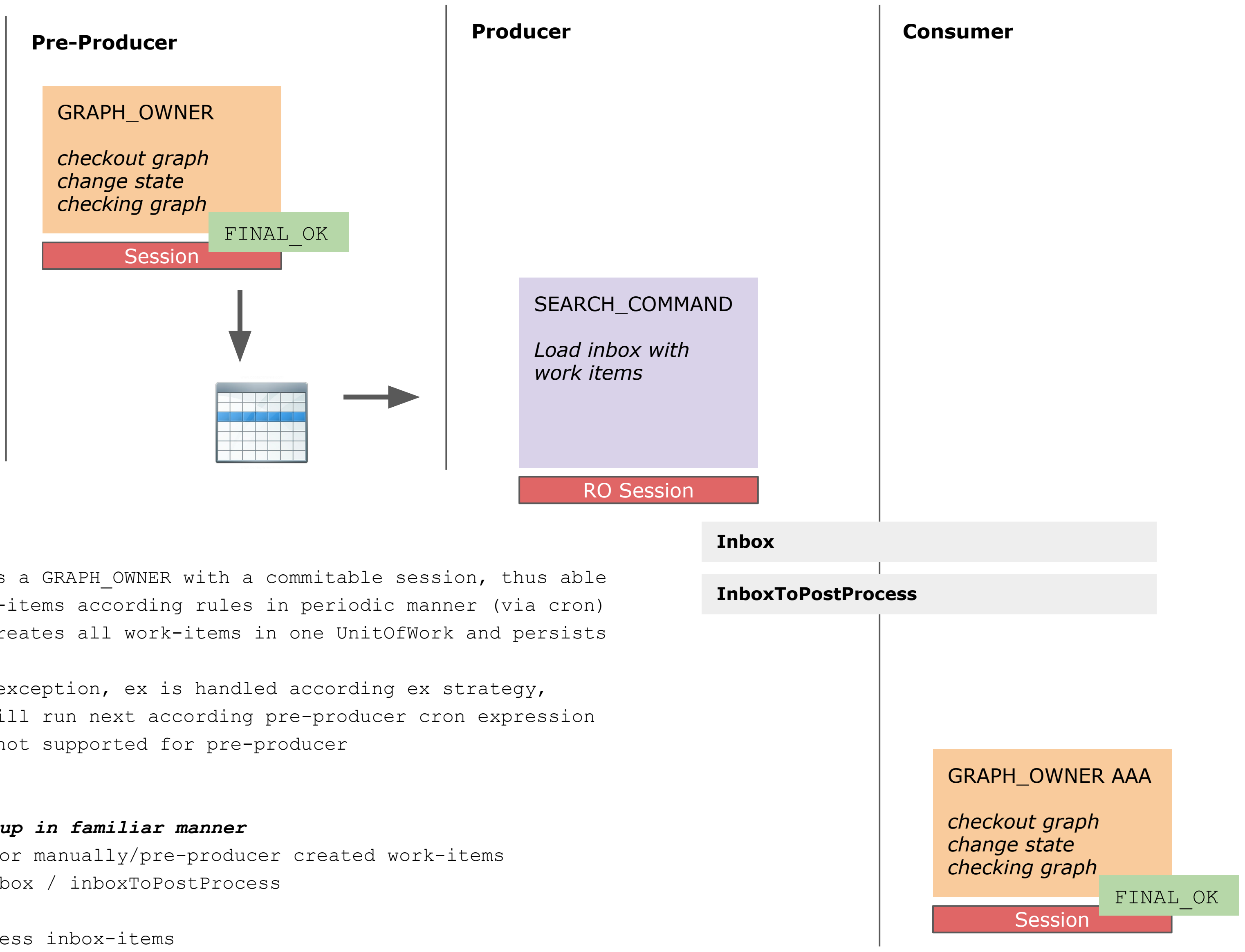
GRAPH_OWNER Command
Create CCC from temporary ViewObject

Producer/Consumer pair 3

SEARCH_COMMAND
Create Temporary ViewObjects with collections of CCCs

GRAPH_OWNER Command
Create DDD from temporary ViewObject

OFXBatchJob Pre-Producer setup



Pre-producer setup

- Pre-Producer is a GRAPH_OWNER with a commitable session, thus able to create work-items according rules in periodic manner (via cron)
- Pre-Producer creates all work-items in one UnitOfWork and persists them
- In case of an exception, ex is handled according ex strategy, pre-producer will run next according pre-producer cron expression
- Delay mode is not supported for pre-producer

Producer/Consumer setup in familiar manner

Producer will check for manually/pre-producer created work-items
Producer will load inbox / inboxToPostProcess

Consumer(s) will process inbox-items

Bear in mind

1. No complex inbox calculation, no complex logic -> no exceptions in producer
2. Do not build batchjobs in dependent mode if not necessary -> more stability

MoWare CMD 2017

Continuous Delivery & Testing

“The **growing code base** must be cleaned regularly during development.”

Why clean code?

“A building with broken windows looks like nobody cares about it. So other people stop caring” (Dave Thomas, Andy Hunt)

“Most Software today is very much like an egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousands of slaves” (Alan Key)

What is clean code?

“Clean code reads like well-written prose” (Grady Booch)

“[...] easy for others to enhance. It has unit and acceptance Test […]” (Dave Thomas)

“(1) Runs all tests, (2) contains no duplication, (3) expresses design ideas, (4) minimizes the number of entities” (Ron Jeffries)

“Without a test suit they lost the ability to make sure that changes to their code base worked as expected [...] their defect rate began to rise [...] they started to fear making changes.” (Clean Code)

“It is the tests that keep code flexible, maintainable and reusable.” (Clean Code)

Prerequisite for Continuous Delivery? And a nice docu?

Continuous Delivery & Testing

Independence

Drop Tables
Create them
Create data
LocalDB / TestDB

Independence

Run tests in any order
Run specific tests
Debug specific tests

Repeatable

Create necessary env.
> Data to throw away
> Establish precond.
> Date/Time Handling
>

Fast

Build a suit of tests
Exec them at once
Run them frequently

Self-Validating

Passed / notPassed
Asserts + desc
Graph Compare +
Visualize
Sunny Day/Rainy Day

?

Model and check
complete Business
use-cases ?

Start developing in the
sandbox?

Refactor

Have a sandbox
Iterate, build a solution
Formulate Tests
Refactor the solution

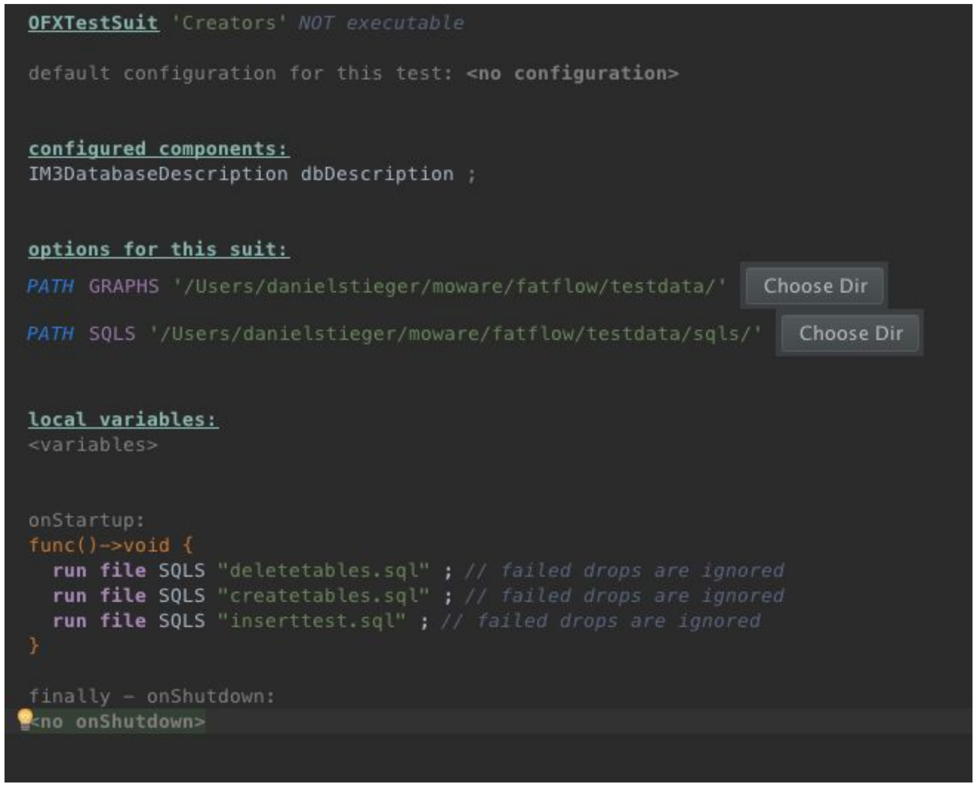
Documentation

Show story (ies)
Show initial data, show
results
Debug and log progress
with specific elems.

I Setup a nice Test-Environment before development

Prepare Test-Environment

> Link necessary Master-Data somehow?
> Delete and re-create important Tables (for project entities)
> Import relevant data (e.g. with insert statements)
> Collect and save important ext. artefacts (e.g. xml files to import)



```
OFXTestSuit 'Creators' NOT executable

default configuration for this test: <no configuration>


configured components:
IM3DatabaseDescription dbDescription ;


options for this suit:
PATH GRAPHS '/Users/danielstieger/moware/fatflow/testdata/'  Choose Dir

PATH SQLS '/Users/danielstieger/moware/fatflow/testdata/sqls/'  Choose Dir


local variables:
<variables>


onStartup:
func()->void {
  run file SQLS "deletetables.sql" ; // failed drops are ignored
  run file SQLS "createtables.sql" ; // failed drops are ignored
  run file SQLS "inserttest.sql" ; // failed drops are ignored
}

finally - onShutdown:
<no onShutdown>
```

II Write Tests

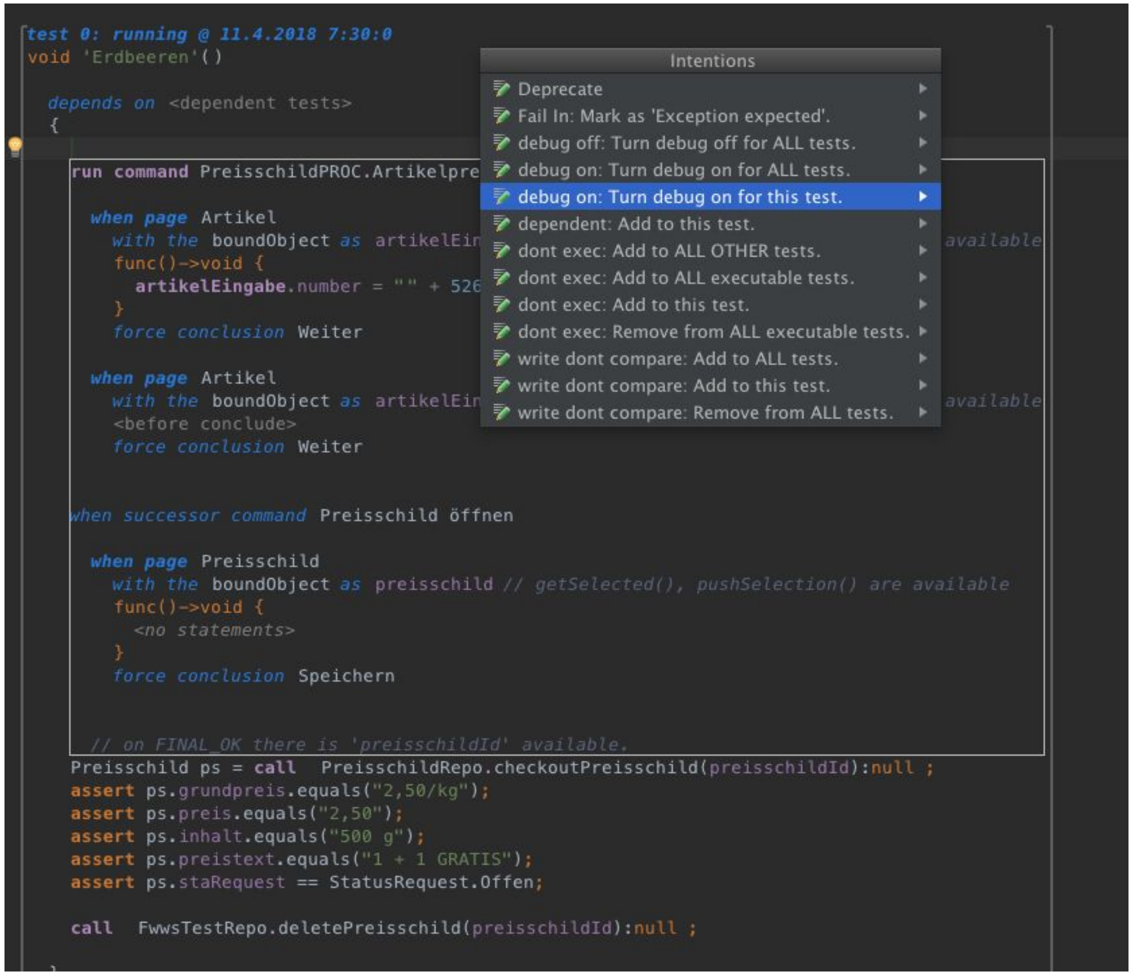
Create Tests

> Each Tests should create its own data
> Tests can use dependent tests to do so
> Tests can focus on small service methods with logic (unit tests)
> Tests can also focus on complex procedures, like Graph_owner with multiple Graph_edits applied (integration tests)
> Use assert statements to check if the tested artefact does, what it should
> Do not use the graph compare functionality (somehow not that cool ..)

>

III Use Tests

> Tests can be run in isolated mode, in order to focus (exec single test only)
> Tests can be run in a special “debug mode”
>


Intentions
Deprecate
Fail In: Mark as 'Exception expected'.
debug off: Turn debug off for ALL tests.
debug on: Turn debug on for ALL tests.
debug on: Turn debug on for this test.
dependent: Add to this test.
dont exec: Add to ALL OTHER tests.
dont exec: Add to ALL executable tests.
dont exec: Add to this test.
dont exec: Remove from ALL executable tests.
write dont compare: Add to ALL tests.
write dont compare: Add to this test.
write dont compare: Remove from ALL tests.

```
test 0: running @ 11.4.2018 7:30:0
void 'Erdbeeren'()

  depends on <dependent tests>
  {

    run command PreisschildPROC.Artikelpre

      when page Artikel
        with the boundObject as artikelEin                                    available
        func()->void {
          artikelEingabe.number = "" + 526
        }
        force conclusion Weiter

      when page Artikel
        with the boundObject as artikelEin                                    available
        <before conclude>
        force conclusion Weiter


    when successor command Preisschild öffnen

      when page Preisschild
        with the boundObject as preisschild // getSelected(), pushSelection() are available
        func()->void {
          <no statements>
        }
        force conclusion Speichern


      // on FINAL_OK there is 'preisschildId' available.
    Preisschild ps = call  PreisschildRepo.checkoutPreisschild(preisschildId):null ;
    assert ps.grundpreis.equals("2,50/kg");
    assert ps.preis.equals("2,50");
    assert ps.inhalt.equals("500 g");
    assert ps.preistext.equals("1 + 1 GRATIS");
    assert ps.staRequest == StatusRequest.Offen;

    call  FwwsTestRepo.deletePreisschild(preisschildId):null ;
```

MoWare CMD 2017

(1) **Command Patterns**

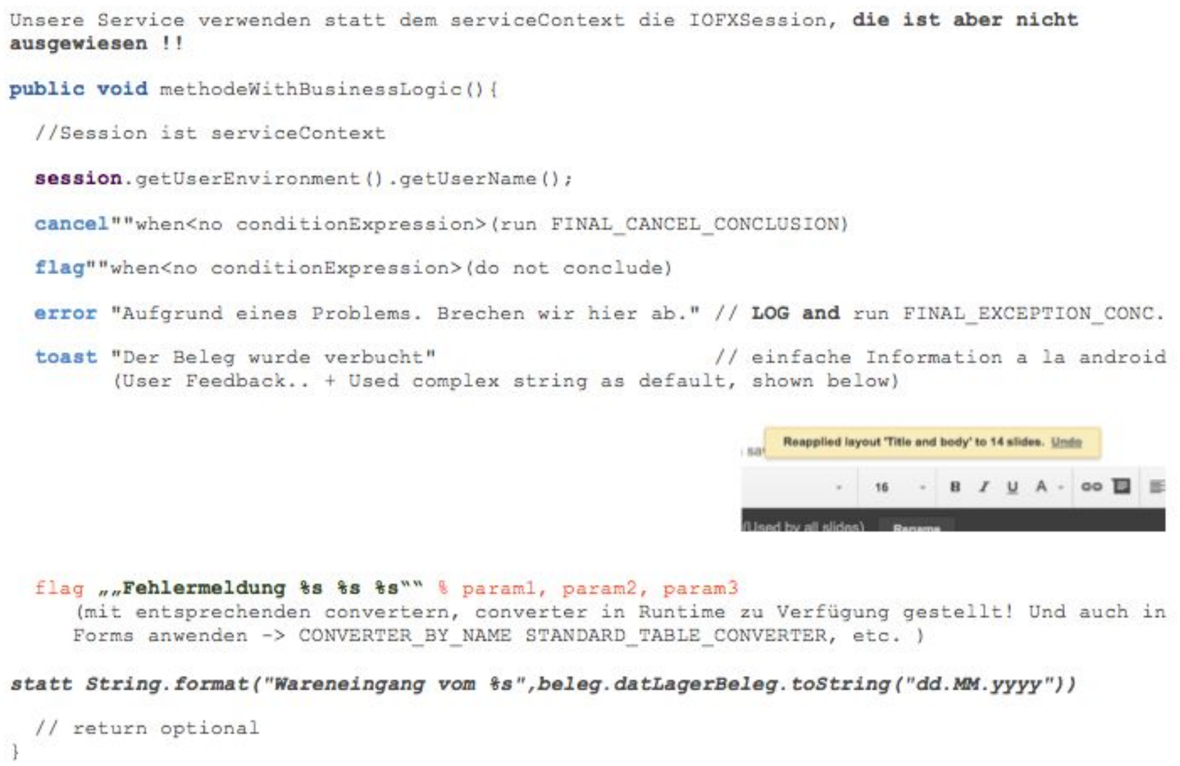
(2) **OFXBatchJobs**

(3) **OFXTestSuit**

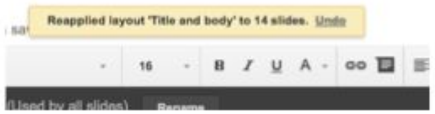
(4) **Additional Features**

(5) **Lessons learned**

Was gehört den eigentlich zur Business-Logik?



```
Unsere Service verwenden statt dem serviceContext die IOFXSession, die ist aber nicht
ausgewiesen !!

public void methodeWithBusinessLogic(){

  //Session ist serviceContext

  session.getUserEnvironment().getUserName();

  cancel""when<no conditionExpression>(run FINAL_CANCEL_CONCLUSION)

  flag""when<no conditionExpression>(do not conclude)

  error "Aufgrund eines Problems. Brechen wir hier ab." // LOG and run FINAL_EXCEPTION_CONC.

  toast "Der Beleg wurde verbucht"                       // einfache Information a la android
        (User Feedback.. + Used complex string as default, shown below)
```



```
  flag „„Fehlermeldung %s %s %s"" % param1, param2, param3
     (mit entsprechenden convertern, converter in Runtime zu Verfügung gestellt! Und auch in
     Forms anwenden -> CONVERTER_BY_NAME STANDARD_TABLE_CONVERTER, etc. )

statt String.format("Wareneingang vom %s",beleg.datLagerBeleg.toString("dd.MM.yyyy"))

  // return optional
}
```

Flag mit längerem "***Hint***" Text?
Was ist denn vermutlich falsch?
Könnte mehrere Gründe haben?

Toast and EMIT: System mit Commands und Events

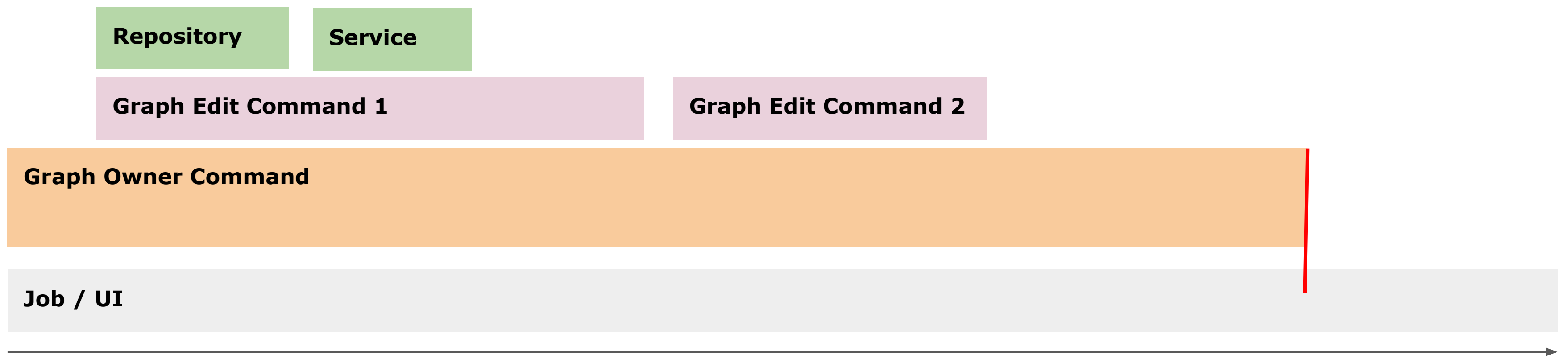
```
toast "Der Artikel wurde geändert"
emit ArticleChanged(param?)

toast "Der Artikel wurde den
Stammdaten hinzugefügt"
emit NewArticleCreated(param?)

toast Um aufzuzeichnen? Wer hat was,
wann gmeacht?
```

Der Service, ein vernachlässigtes Stiefkind? (Domain Service?)
Erzeugen eines Graphen
Transformieren eines Graphen in einen anderen
Graph in sich anpassen (Gesamtsummen etc., D.h. Regeln anwenden)
Prüfen/Validation eines Graphen und dessen Stati anpassen

=> arbeiten mit flag, cancel, error, toast und Texten!
=> what else? Methoden Benennung?
=> Service Architecture & DataStructures vs. Objects & OO Paterns

Commands, Cancellation and Jobs

Statement	Verwendbar	Auswrikung
cancel	Command, Service, Repository	***Aktuelles Command*** in Final_CANCEL beenden, > kein Fehler > in UI mit Meldung an Benutzer > in Job ohne LOG
flag	Command, Service, ~~Repository~~	Unterbrechung der Ausführung in ***aktueller Seite*** > Meldung an Benutzer > Abbruch des Command-Stack im Job + Log
error	Command, Service, Repository	Alle Commands des aktuellen ***Command-Stack*** **(bis zum ersten Graph_Owner)** mit Final_EX beenden > Meldung an Benutzer ("am System nicht ausgeführt werden") > Im Job Log **> Abbruch des aktuellen Commands + Graph_Owner, kein Abbruch des Jobs und der Search Commands.**
toast	Command	Keine Auswirkung auf Ausführung, Erfolgsmeldung(en) aufzeichnen > Meldung an Benutzer > Im Job Log auf Info Niveau / JMX-Message > Systemweit Event auslösen? "new_article id 10" **> nur bei erfolgreicher Transaktion !**
done	Command	Command in Final_OK beenden
page	Command	Seite in Command (Wizzard) wechseln



‘Hello %d World’

The ` ‘New String‘ ` Implementation

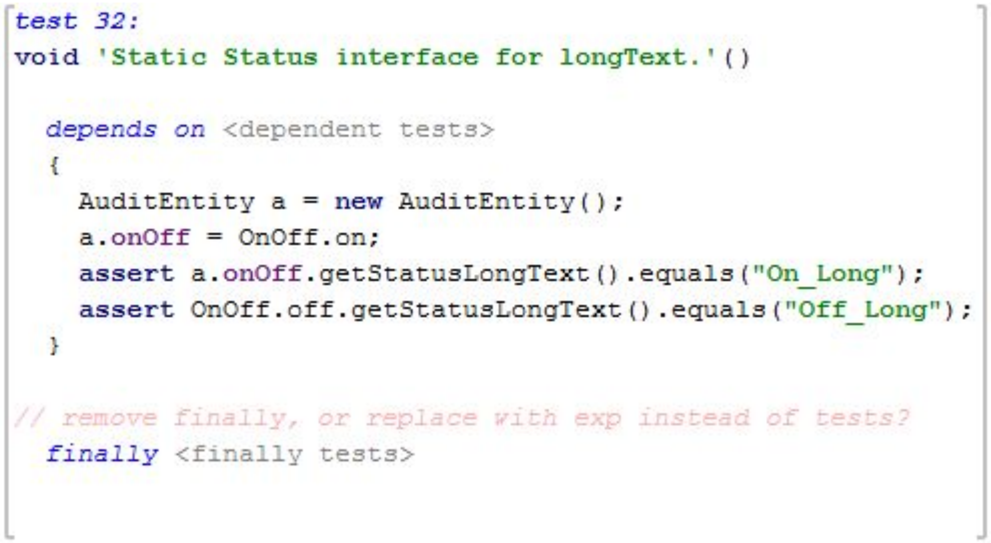
Expression		
%d	integer	String.format
%s	string	String.format
%bd	BigDecimal	
%ld	LocalDate	
%dt	DateTime	
%st	Status	Status Short Text
%obj	“Object”	Applying toString() if arg not null
%%		Escape % char

Fehlt: Laufzeitunterstützung zur Verwendung von MultiString in Entitäen

MultiStringImplementation.format()

*Singleton Pattern + **Abgrenzung mit () bei den Parametern***

Status Handling



```
test 32:
void 'Static Status interface for longText.'()

  depends on <dependent tests>
  {
    AuditEntity a = new AuditEntity();
    a.onOff = OnOff.on;
    assert a.onOff.getStatusLongText().equals("On_Long");
    assert OnOff.off.getStatusLongText().equals("Off_Long");
  }

// remove finally, or replace with exp instead of tests?
  finally <finally tests>
```

> #Meta Informationen only available for properties
- setEnabled()
- setOptional()
- ***setLabel()***
- requestFocus()

> **Callable at status**: getStatusLongText(), getStatusShortText(), getStatusDBText()

Services and Slicing

> Service für spezifische Geschäftslogik statt Service für spezifisches Dokument
> Geschäftslogik verschiedener Dokumententypen in einem Service zusammengefasst (statt ein Service pro Dokumententyp)
> “Objektorientierte Unterstützung” Dynamische Delegation zur Laufzeit durch ServiceMethode(n)

Service Verbuchen

Retoure
Wareneingang
VerkaufKassa
Umbuchung

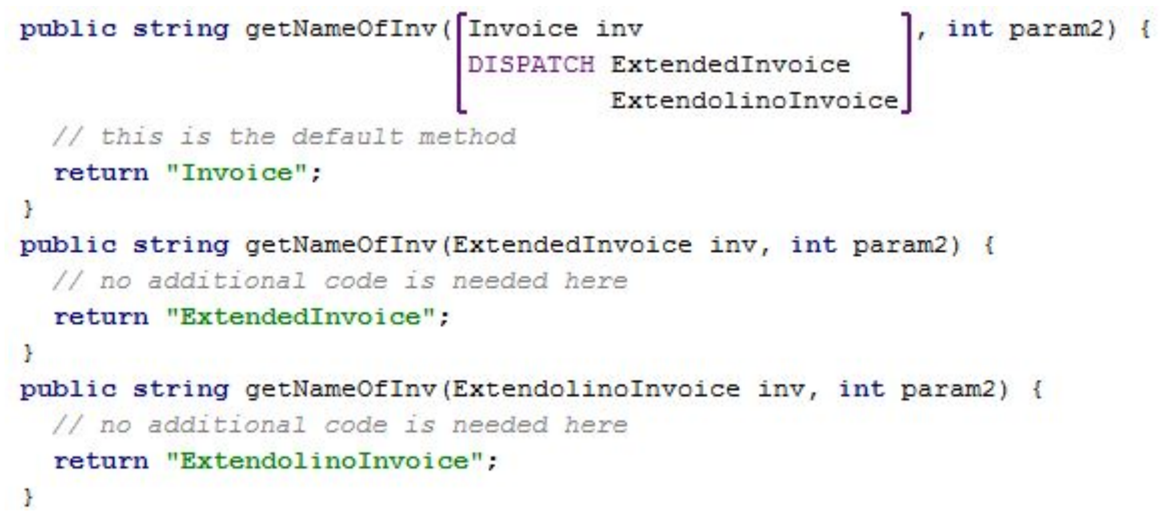
Service Validation

Retoure
Wareneingang
VerkaufKassa
Umbuchung

Service UILabelsAndTitles

Retoure
Wareneingang
VerkaufKassa
Umbuchung

Dispatching



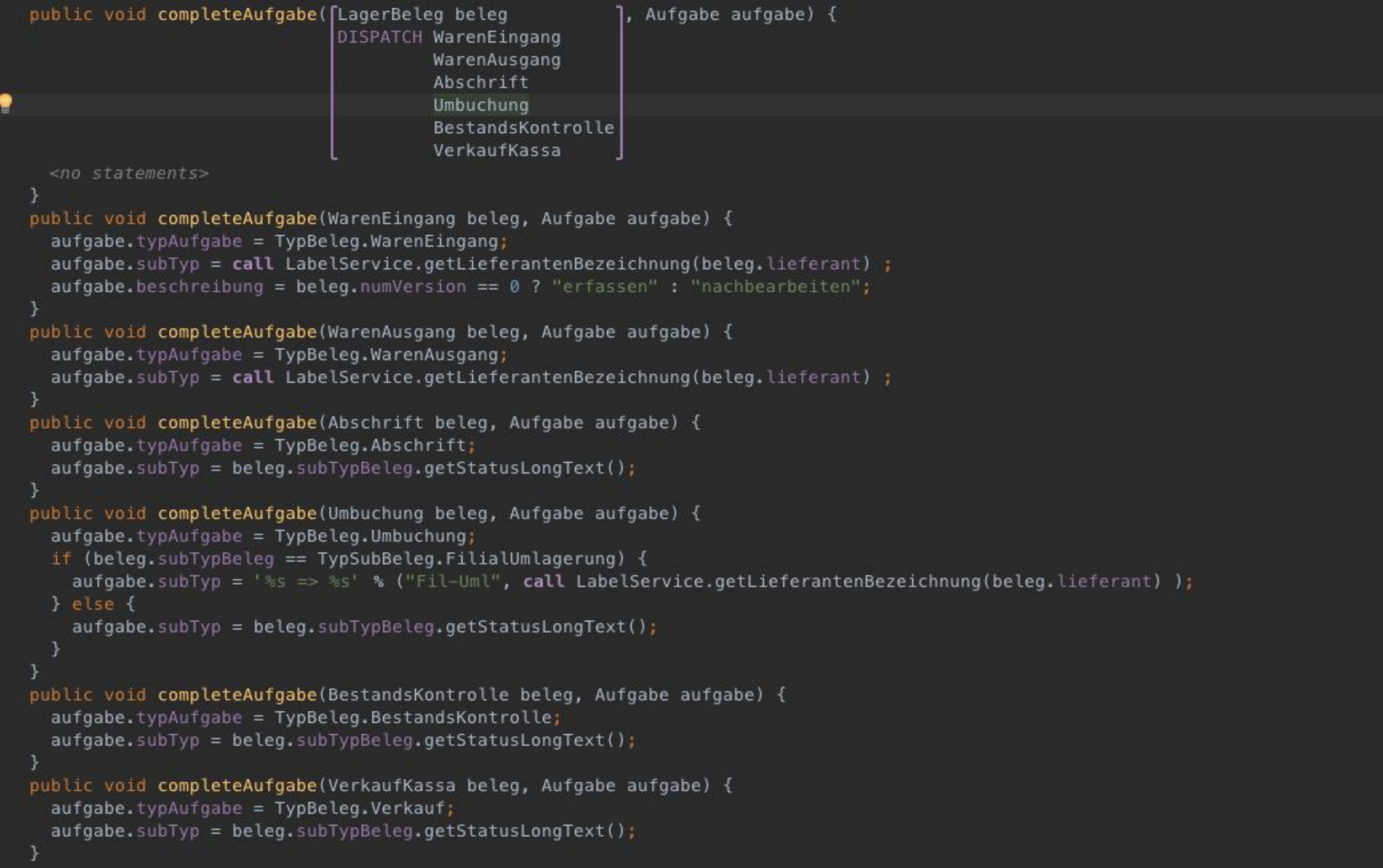
```
public string getNameOfInv([Invoice inv                 ], int param2) {
                           DISPATCH ExtendedInvoice
                                    ExtendolinoInvoice]
  // this is the default method
  return "Invoice";
}
public string getNameOfInv(ExtendedInvoice inv, int param2) {
  // no additional code is needed here
  return "ExtendedInvoice";
}
public string getNameOfInv(ExtendolinoInvoice inv, int param2) {
  // no additional code is needed here
  return "ExtendolinoInvoice";
}
```

> Method overwriting at runtime: available only in Services

> Call Method, which forwards to more specific method regarding the dispatched param

> Provide a default method

> extensive checking by dispatch attribute



```
public void completeAufgabe([LagerBeleg beleg          ], Aufgabe aufgabe) {
                            DISPATCH WarenEingang
                                     WarenAusgang
                                     Abschrift
                                     Umbuchung
                                     BestandsKontrolle
                                     VerkaufKassa
  <no statements>
}
public void completeAufgabe(WarenEingang beleg, Aufgabe aufgabe) {
  aufgabe.typAufgabe = TypBeleg.WarenEingang;
  aufgabe.subTyp = call LabelService.getLieferantenBezeichnung(beleg.lieferant) ;
  aufgabe.beschreibung = beleg.numVersion == 0 ? "erfassen" : "nachbearbeiten";
}
public void completeAufgabe(WarenAusgang beleg, Aufgabe aufgabe) {
  aufgabe.typAufgabe = TypBeleg.WarenAusgang;
  aufgabe.subTyp = call LabelService.getLieferantenBezeichnung(beleg.lieferant) ;
}
public void completeAufgabe(Abschrift beleg, Aufgabe aufgabe) {
  aufgabe.typAufgabe = TypBeleg.Abschrift;
  aufgabe.subTyp = beleg.subTypBeleg.getStatusLongText();
}
public void completeAufgabe(Umbuchung beleg, Aufgabe aufgabe) {
  aufgabe.typAufgabe = TypBeleg.Umbuchung;
  if (beleg.subTypBeleg == TypSubBeleg.FilialUmlagerung) {
    aufgabe.subTyp = '%s => %s' % ("Fil-Uml", call LabelService.getLieferantenBezeichnung(beleg.lieferant) );
  } else {
    aufgabe.subTyp = beleg.subTypBeleg.getStatusLongText();
  }
}
public void completeAufgabe(BestandsKontrolle beleg, Aufgabe aufgabe) {
  aufgabe.typAufgabe = TypBeleg.BestandsKontrolle;
  aufgabe.subTyp = beleg.subTypBeleg.getStatusLongText();
}
public void completeAufgabe(VerkaufKassa beleg, Aufgabe aufgabe) {
  aufgabe.typAufgabe = TypBeleg.Verkauf;
  aufgabe.subTyp = beleg.subTypBeleg.getStatusLongText();
}
```

MoWare NI 2017

Elements in a ViewObject ListMember can be exchanged with Selection Updates in Search Command Pattern.

Decoupling of Parent Commands form CommandContainers.

Color Management with Stati and Page Panes

0 and Table Format for “no0”

Dynamic Tiles

Nullable queries with manmap

Shouldn’t we cancel a GraphOwner when we encouter an exception in an graph edit? Typically Yes !

Same Semantics as in Batch Job?

MoWare 2017

Elements in a ViewObject ListMember can be exchanged with Selection Updates in Search Command Pattern.

Decoupling of Parent Commands from CommandContainers.

Color Management with Stati and Page Panes

0 and Table Format for “no0”

Dynamic Tiles

Nullable queries with manmap

Shouldn’t we cancel a GraphOwner when we encounter an exception in an graph edit?
Typically Yes !

Same Semantics as in Batch Job?

Conversion Forms3 auf DataUX

> alle notwendigen Branches mergen, dann konvertieren

> extended Doc in: Moware Supplemental Documentation

MoWare CMD 2017

Anti Patterns for Document Centered Applications

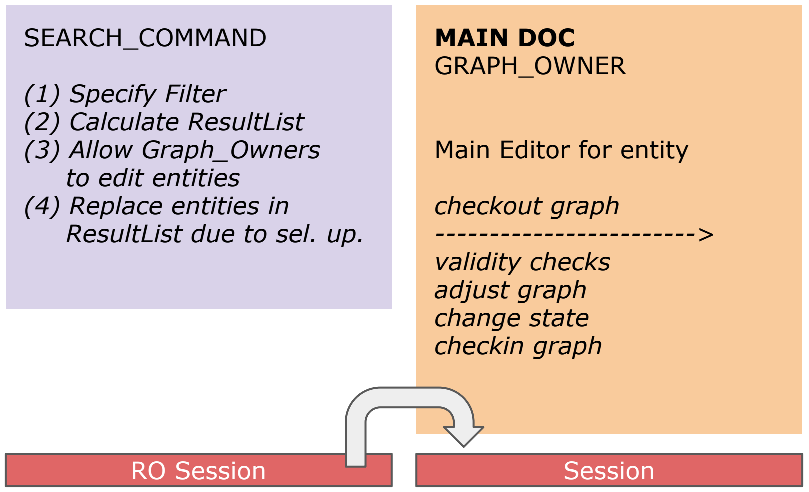
> User Perspective: various apps, same handling, same style, same expectations

> Developer Perspective: assumptions regarding organization of functionality, expectations, maintenance and change (even own code)

> Search and find bugs in other SW (graph debugger, tests)

Search / Main Doc / Graph-Edit distinction

> do not mix up Search as visualization for editing and
> main doc for editing


SEARCH_COMMAND

(1) Specify Filter
(2) Calculate ResultList
(3) Allow Graph_Owners to edit entities
(4) Replace entities in ResultList due to sel. up.

MAIN DOC
GRAPH_OWNER

Main Editor for entity

checkout graph
------------------------->
validity checks
adjust graph
change state
checkin graph

Most importantly: ***unit of work*** *gets diluted !*

What is the unit of work - what has to be consistent when multiple users are editing the same unit of work? (e.g. locking! And update on checkin!)

Not only relevant for ui standardization, ***user understanding,*** *but also for performance / mem usage on server env.*

How to handle things:
> main doc is for editing visualization ***(no enabled delegates)***
> various graph edits is for editing
> search will be updated via update selection of main doc
> clear separation of search command entities and graph in main doc

> The special Case? No graph available? SGO Pattern?