# ESPRIT Project N. 25 338

## Work package I

### Pilot Application 1

# Design

| | | | |
|---|---|---|---|
| ID: | WP_I_Design V. 0.1 | Date: | 15.05.1998 |
| Author(s): | Hans-Guenter Stein, FAST e.V. | Status: | draft |
| Reviewer(s): | | Distribution: | Project internal |

# Change History

| Document Code | Change Description | Author | Date |
|---|---|---|---|
| WP_I_Design | Version 0.1 | Stein, | 22.06.98 |

# Overview

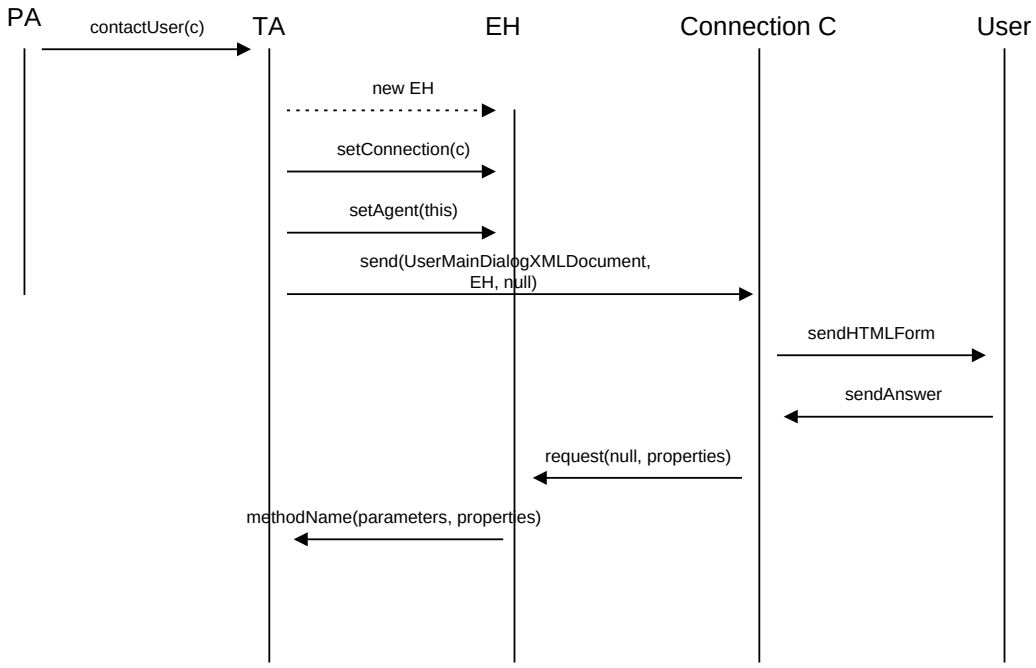
Design issues are discussed here serve as implementation guideline for pilot application programmers. The document's content is highly dynamic as it evolves with the ongoing implementation and design iteration process. At the current stage the document provides enough detail to guide the programmers through the implementation of a first prototype of the applications. The intention was to provide as much detail as possible with respect to the application's core component whereas 'add-ons' not essential for the first prototype implementation are left open or are only briefly addressed. Therefore the document contains lots of annotations raising issues not yet solved in all details. These annotations serve as remainders for future design iterations.

# 1 Design for PA 1

## *1.1 User Interfaces*

Derived from the use case descriptions in the requirement document, the user interfaces that are required for the two pilot services along with the functionality encapsulated in them are listed in the following section.

The following diagram illustrates how agents communicate with a User Access Connection via a generic event handler. For more details on this issue see the design documents of WP D and H.




**General Annotations on issues to be addressed in future versions:**

- Various help pages are needed to explain the use of the forms.
  - When to display them?

  - How to display (-> separate windows preferred)?
  - When to close them (-> when the referring form is closed)?
- How to handle ‘back’ and ‘reload’ requests from the browser?
- Exception and confirmation messages are needed every now and then.
- When objects are copied out of IS, the objects in IS must get locked for further write access (in most situations). This issue has already been addressed in some detail in the design of the interfaces.
- All XSL-forms that are static and will be used by all users should be stored in a central repository accessible by all users’ TAs
- Where is ‘state’ required?
  - When interacting with users, all transactions should be asynchronous. This makes dealing with user interaction via a stateless browser more easy to handle.
- Integration of components delivered by other partners which are not available at present.
  - How does communication flow among UA, PA, TA, IS, Profile, Diary and external Java classes?

### 1.1.1 PA specific user interfaces

Here we describe the user interfaces that need to be provided by the Personal Assistant component. These forms are not application specific and therefore are part of the design and implementation of UWE’s agent framework. They will not be discussed in greater detail in this document.

**FollowMe entry page (from a web server)**

- displays a form to enter a PA name
- functions:
  - submit: connect to the PA. This involves looking up the PA by name from some PA directory service and then calling the function PA.contactUser(connection c).

**PA authentication**

- displayed after contactUser(connection c) was called from FM entry point
- display login dialog
- functions:
  - cancel: PA.endSession (see below)
  - login: PA.login
    - verifies user authentication
      - if successful: PA.mainPage
      - if failed: PA.autenticationFailure

**PA authentication failure**

- diplayed after authentication failed
- functions:
  - try again: PA.login
  - cancel: FollowMe entry page

**PA main**

- displayed after successful authentication (login)
- functions:
  - retire from FollowMe: PA.destroy
    - calls confirmation page
  - quit session: PA.endSession
    - ends interactive session; closes the active connection to UA; displays FM entry page
  - manage profile: see below
  - manage services: see below

**PA delete confirmation**

- destroys the PA; requires destroying all things related to this specific PA (IS, TAs,...)
- displayed after PA.destroy was called
- displays a confirmation page to delete the PA
- functions:
  - confirm: delete PA and back to FM entry point page
  - cancel: back to PA main page

**Profile manager main**

- displayed when PA.maintainProfile is called
- functions:
  - back to PA main page
  - maintain Personal Profile
    - User must store his location via GIS interaction
  - maintain Personal Diary
    - User must store at least one contact address (see UA connectivity description)

**Listing of services**

- displayed after PA.maintainServices is called
- displays list of currently active TAs and a list of available new TAs that are not yet in use (missions to be used; namely stock portfolio service + regional event notification service)
- functions:

- new service: MissonServer.getMissionDescription
- back to PA main page
- select active TA: TA XXX contact page

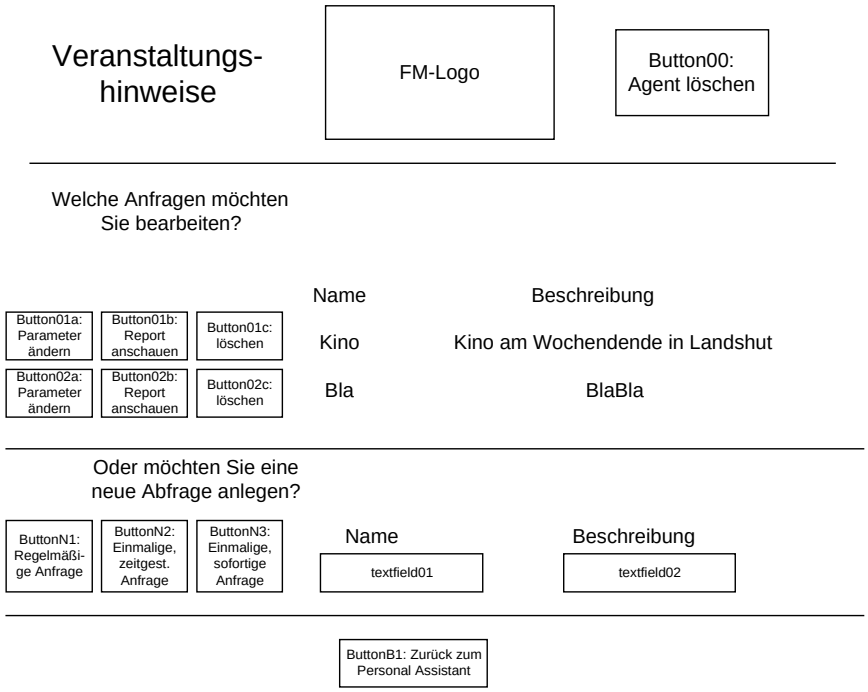
**Service description**

- displayed after a specific mission was selected from the listing of services page
- functions:
  - accept: createAgent(mission) and display TA contact page
  - cancel: back to listing of services (PA.maintainServices)

## 1.1.2 FMEventNotification specific user interfaces

**TA FMenvents contact page**

- displayed either after new TA was created or existing TA was selected from list in listing of services page (or after a ‘back to contact page’ was issued from some other page)
- functions:
  - delete TA: TA.delete
  - change parameters by selecting one entry from the list of defined queries
  - check available data and provide immediate report
  - delete query from list of defined queries: display same page with refreshed list of defined queries
  - create repeated scheduled query
  - create one time scheduled query
  - create one time query with immediate reporting
  - back to listing of services
  - disconnect from FollowMe



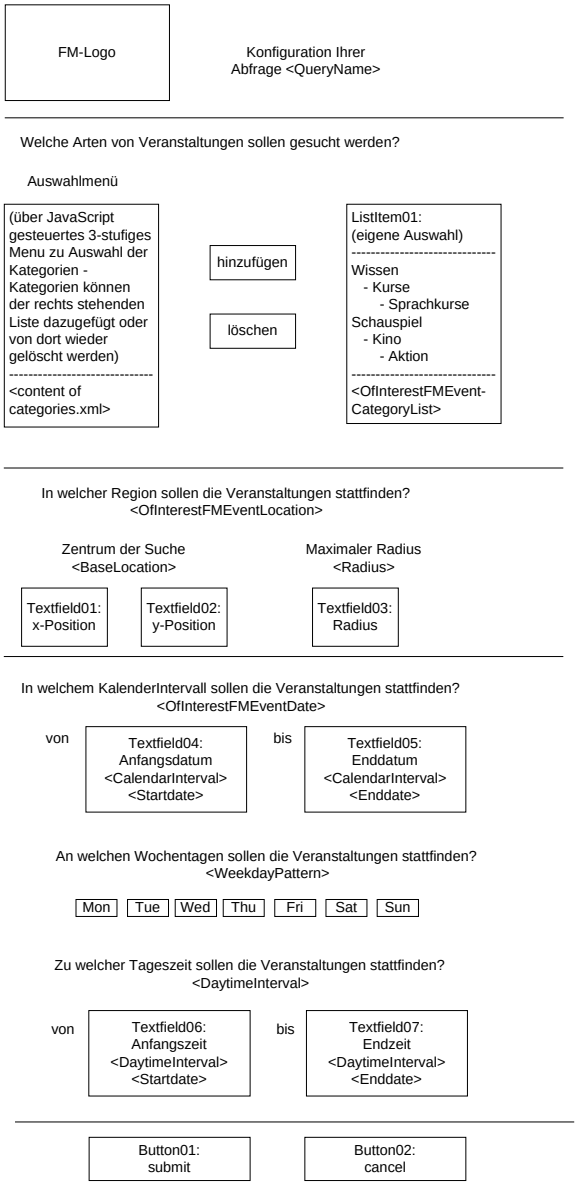

**TA delete confirmation**

- displayed after TA.delete was called
- displays a confirmation page to delete the TA
- functions:
  - confirm: delete TA and back to listing of services
  - cancel: back to FMEvents contact page

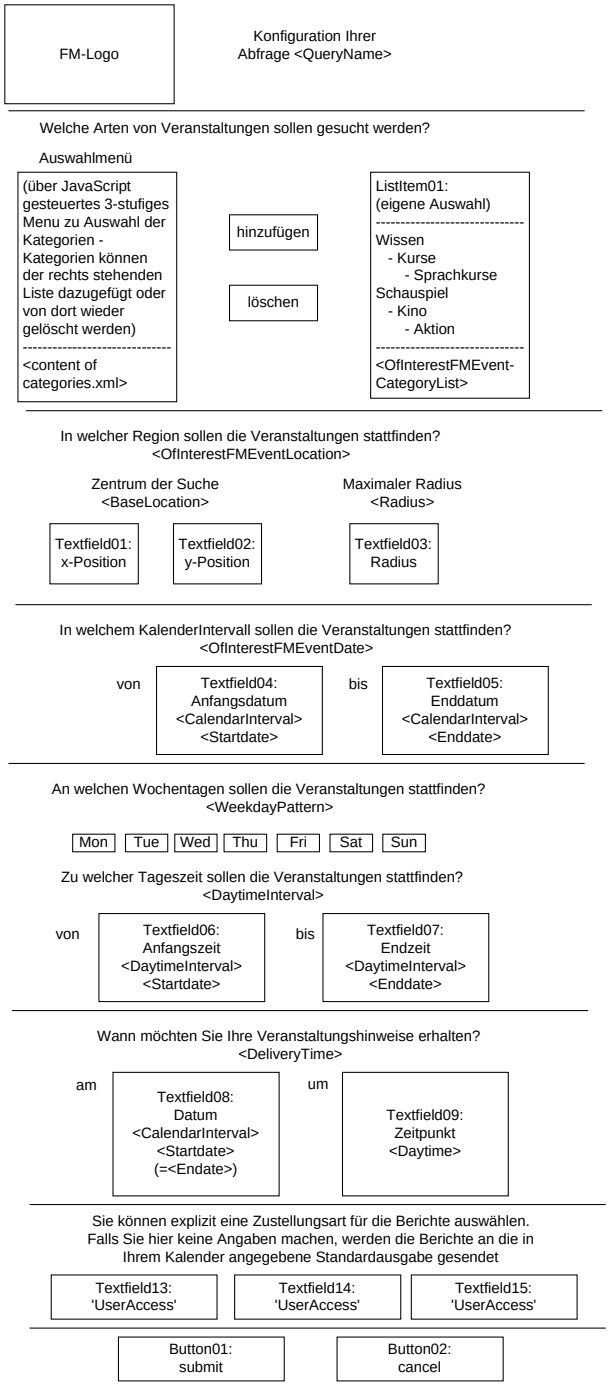
**TA FMEvents set parameters page**

- displayed after the following events (from TA FMEvents contact page)
  - change parameters by selecting one entry from the list of defined queries
  - create repeated scheduled query
  - create one time (scheduled or immediate) query (maybe a shorter form with no scheduling parameters)
- display form for query parameter input
- functions:
  - submit parameters: TA.getParameters
  - definition of location via GIS interaction (default location derived from Personal Profile)
  - cancel: back to TA FMEvents contact page (if a new TA was created and cancel is pressed, the new TA must be deleted!!)

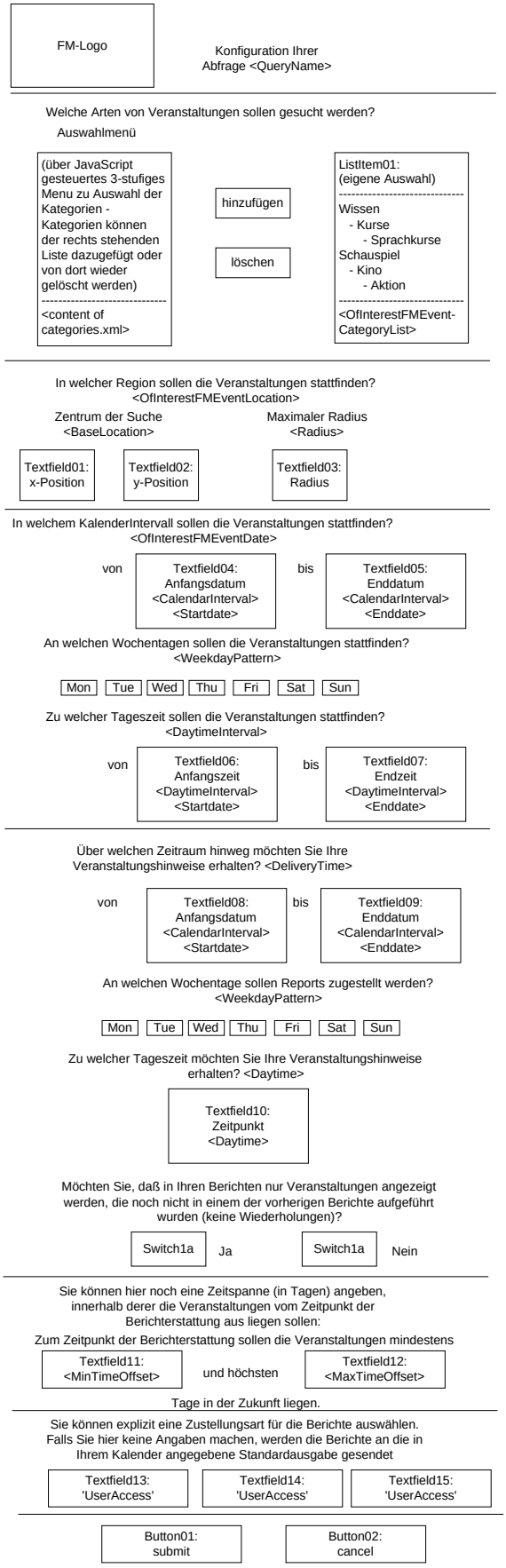
The layout of this page depends on the type of query as illustrated in the following diagrams:




query parameter input form in case of a one time, immediately executed query



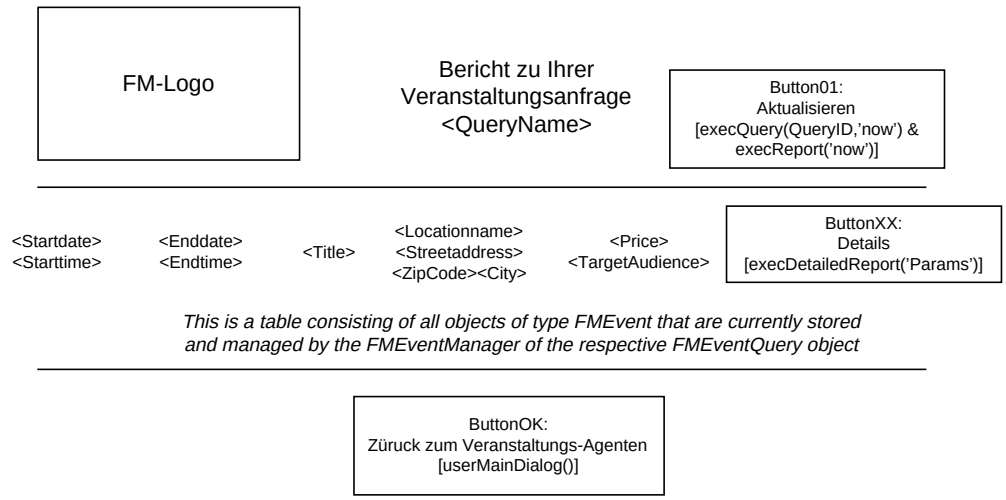

query parameter input form in case of a one time, scheduled query




query parameter input form in case of a regular, scheduled query

**TA FMEvents report page**

- displayed after the following events (from TA FMEvents contact page)
  - check available data and provide immediate report
  - get new data and deliver report immediately
  - create one time query with immediate reporting
- delivers a report on stored event information to the users browser
  - report consists of a list of all events stored under the respective QueryID
- functions:
  - back to TA FMEvents contact page
  - refresh data: triggers immediate collection of data
  - detailed report: by selecting a specific entry from the list of all events, more information on that events is displayed




*In fax-mode (mode = statichtml): leave out all buttons*
*In e-mail mode (mode = acsii): print each of above table fields in separate lines and devide events by a line of dashes*
*In sms-mode (mode = shorttext): just state 'FollowMe Veranstaltungs-Agent: Neue Veranstaltungshinweise eingetroffen'*

FMEvents report page

**TA FMEvents detailed report page**

- displayed when an entry from the list of events was selected
- displays detailed information on a specific event
- functions:
  - open a new window with connection to the GIS server

    - (Javascript) newWindow
    - when this window is closed, control is back to the original window
    - no values need to be transferred from GIS to the FM system
  - back to the TA FMEvents report page
  - back to FMEvents contact page

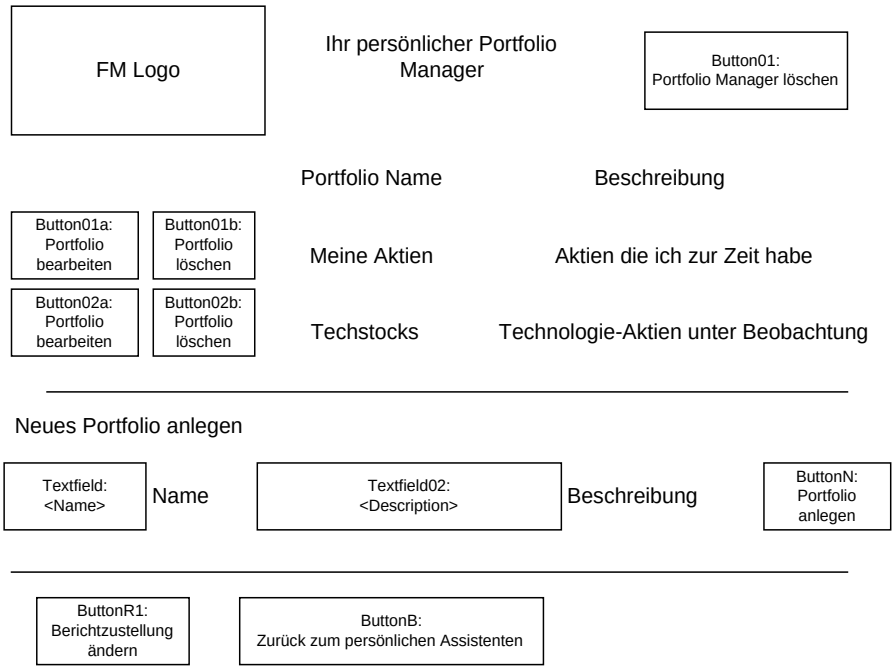
**GIS interaction window**

- displayed when requested from a detailed event description, set query parameters or maintain personal diary pages
- displays a zoomable map of the surroundings of the event location allowing the user to find his way to the event
- on window close: passes control back to the respective TA page and hands over the coordinates where appropriate

## 1.1.3 FMStockManager specific forms and pages

**TA FMStocks contactUser page**

- displayed either after new TA was created, an existing TA was selected from list in listing of services page or after ‘back to portfolio listing’ was called from a specific portfolio page or from the ‘set reporting schedule’ page
- functions:
  - delete TA: TA.delete
  - delete a specific portfolio
  - manage a specific portfolio
  - create a new portfolio
  - change settings for scheduled reporting
  - back to the Personal Assistant (contactUser page or listing of services page?)
  - disconnect from FollowMe



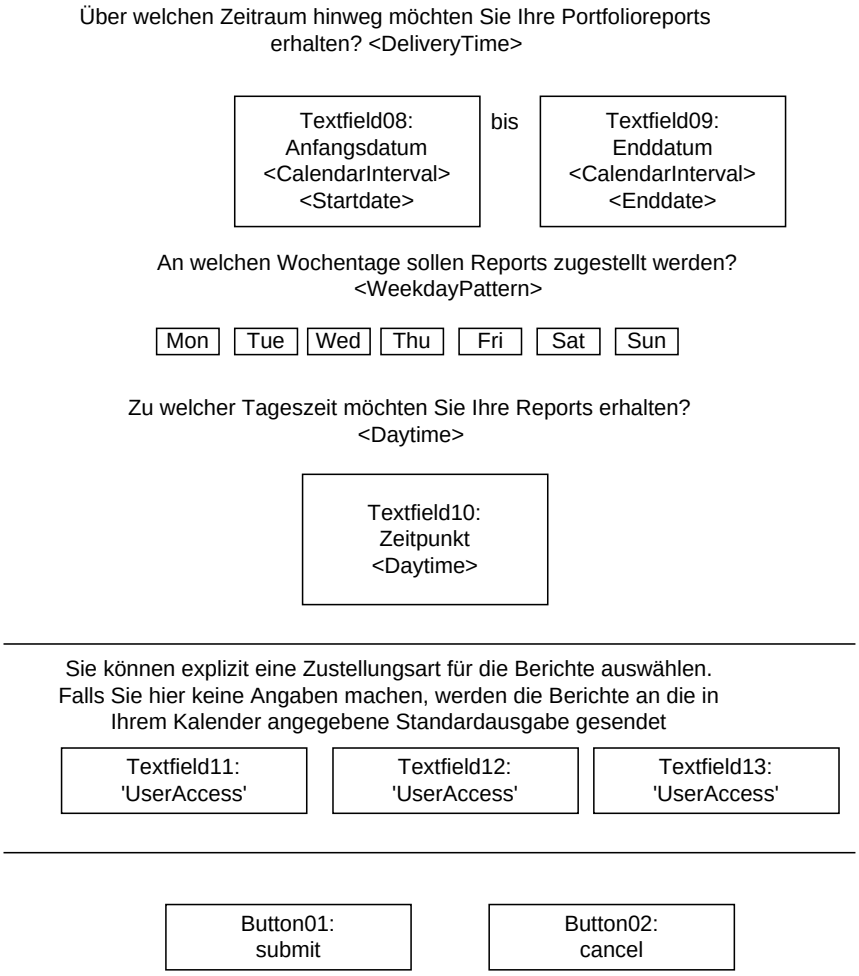

TA FMStocks contactUser page

**TA delete confirmation**

- displayed after TA.delete was called
- displays a confirmation page to delete the TA
- functions:
  - confirm: delete TA and back to listing of services
  - cancel: back to FMStocks contact page

**TA FMStocks set reporting schedule page**

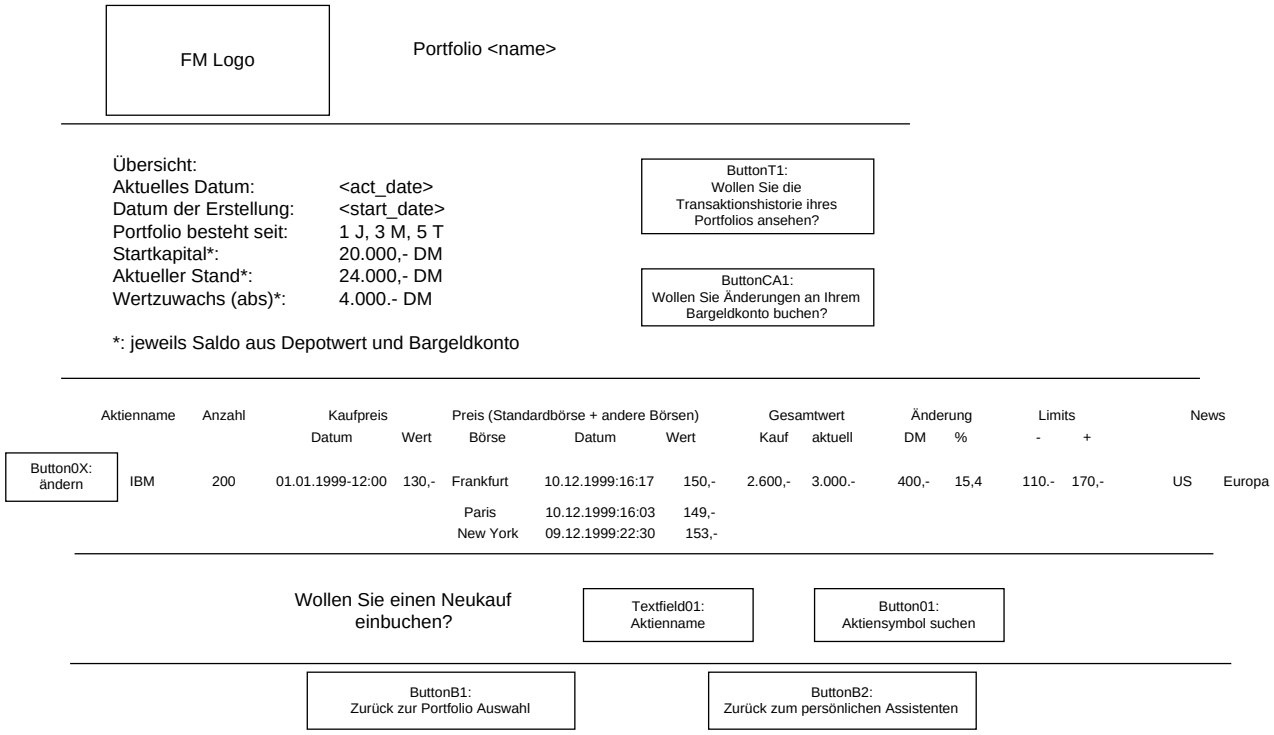
- displayed after ‘change settings for scheduled reporting’ was called from the contactUser page
- display form for reporting schedule parameter input
- functions:
  - submit parameters and back to contactUser page
  - cancel: back to contactUser page (if a new TA was created and cancel is pressed, the new TA must be deleted!!)




TA FMStocks set reporting schedule page

**TA FMStocks manage portfolio page**

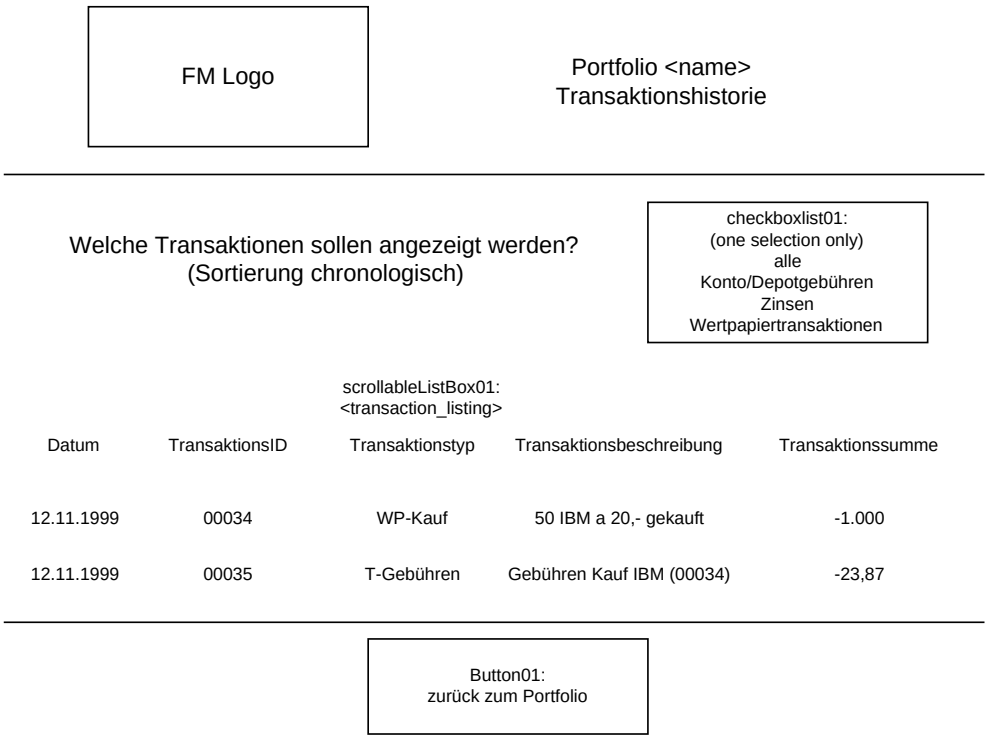
- displayed after ‘manage a specific portfolio’ was called from TA FMStocks contactUser page or after ‘back to portfolio’ was called from ‘manage cash account’, ‘view transaction history’, ‘buy new shares’ or ‘change/delete a position’ pages
- delivers a view on a specific portfolio; share values are downloaded from the local value server to provide up-to-date information
- functions:
  - back to TA FMStocks contactUser page
  - change/delete a specific position in the portfolio
  - buy new shares (= add a new position)
  - open a separate window with industry news related to a specific position (links to external pages at information provider sites)
  - view transaction history
  - manage cash account
  - back to list of portfolios
  - back to PA




TA FMStocks manage portfolio page

## TA FMStocks transaction history page

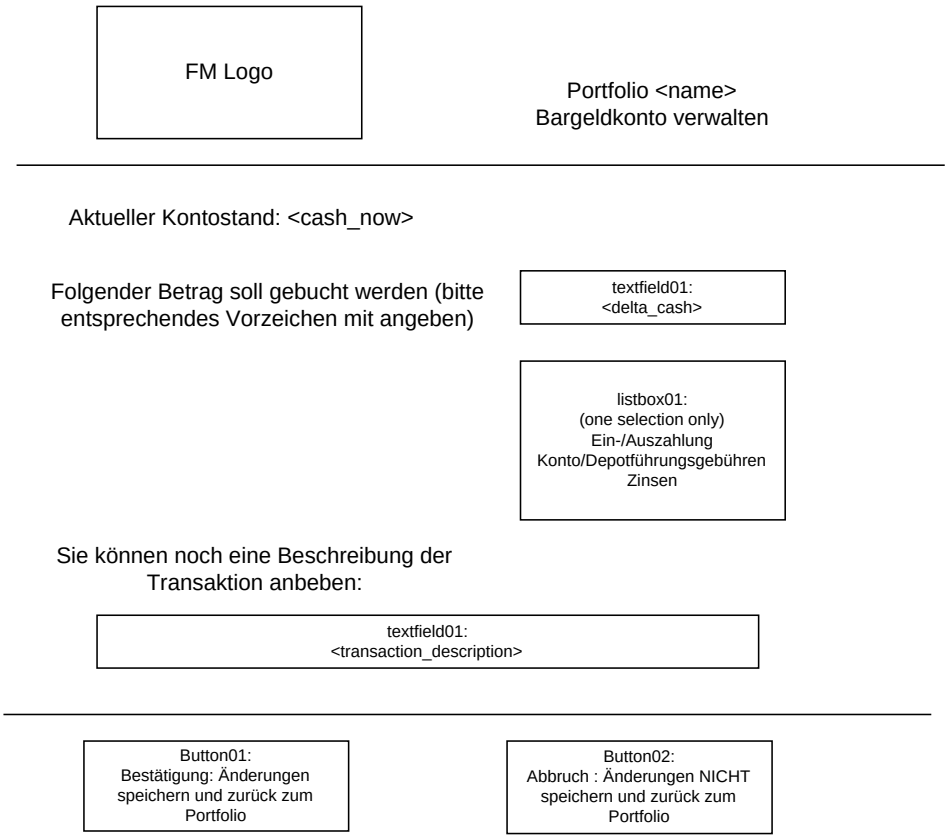
- displayed after ‘view transaction history’ was called from a specific portfolio page
- displays the transaction history for a specific portfolio ordered by date
- functions:
  - back to portfolio
  - select view: ‘all transactions’, ‘only account maintenance costs’, ‘only transactions’,...




TA FMStocks transaction history page

**TA FMStocks manage cash account page**

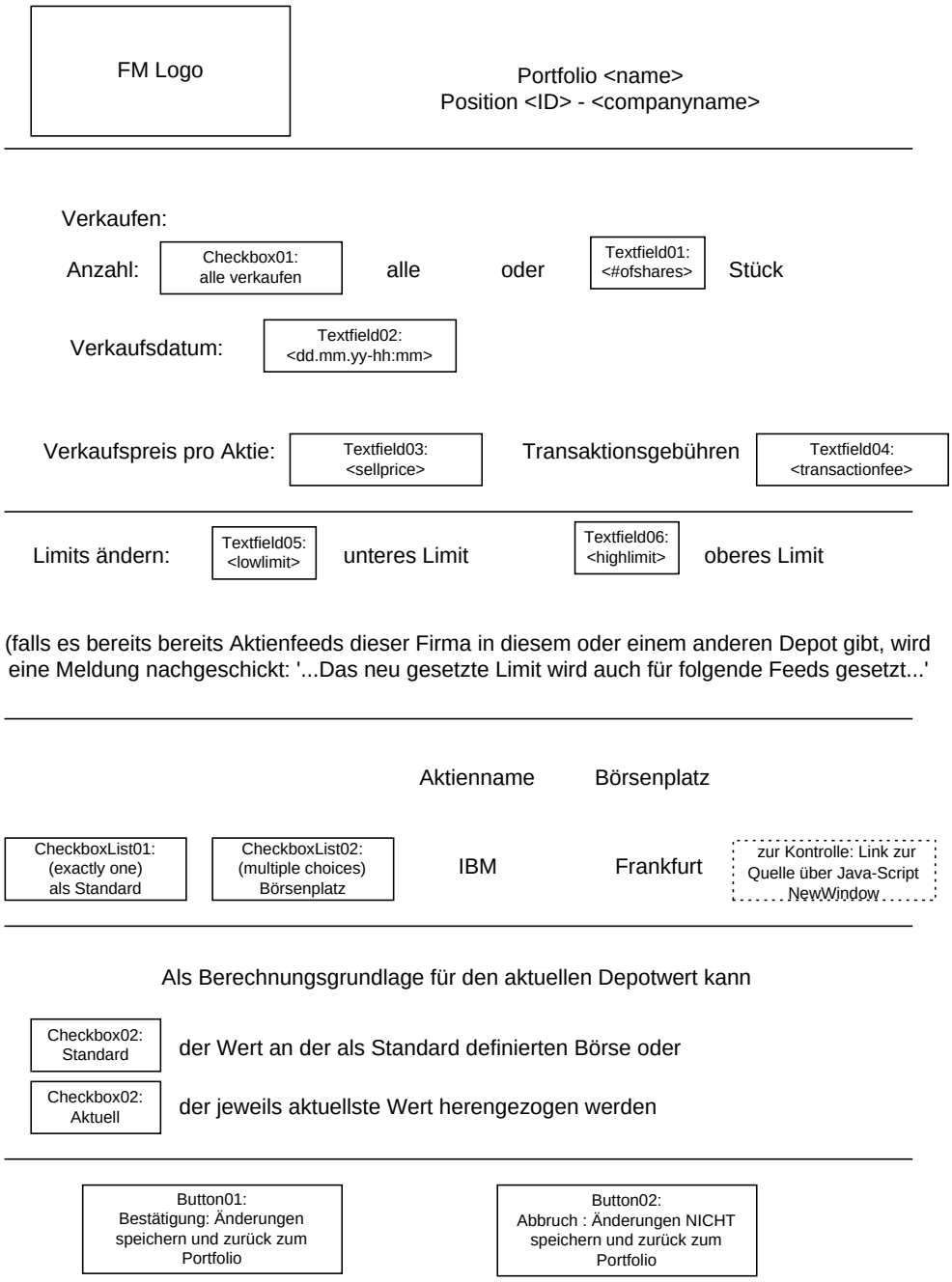
- displayed after ‘manage cash account’ was called from a portfolio page
- allows the user to manually enter changes to the cash account (add/withdraw money, specify maintenance costs)
- functions:
  - enter details on amount of transaction, transaction type and transaction textual description
  - save transaction and back to portfolio page
  - cancel transaction and back to portfolio page




TA FMStocks manage cash account page

**TA FMStocks change/delete position page**

- displayed after ‘change/delete position’ was called from a portfolio page
- allows the user to either sell the whole position, sell parts of the position, change limits for the position or change the sources of values (feeds) the user is interested in
- functions:
  - enter parameters for selling shares
  - enter paramters for adjusting limits
  - display a list of available sources (feeds for the shares)
    - the user may select or de-select one or more sources in the list
    - exactly one source must be selected to act as the default source
    - attached to each source is an link to external pages at value provider sites to allow the user to verify if a source feeds exactly the values the user is interested in; these links are opened in new browser windows
  - the user may specify the sources that should be used to compute the actual value of their shares; options are: ‘always use the default source’ and ‘always use the most up-to-date source’
  - save changes and back to portfolio
  - delete changes and back to portfolio

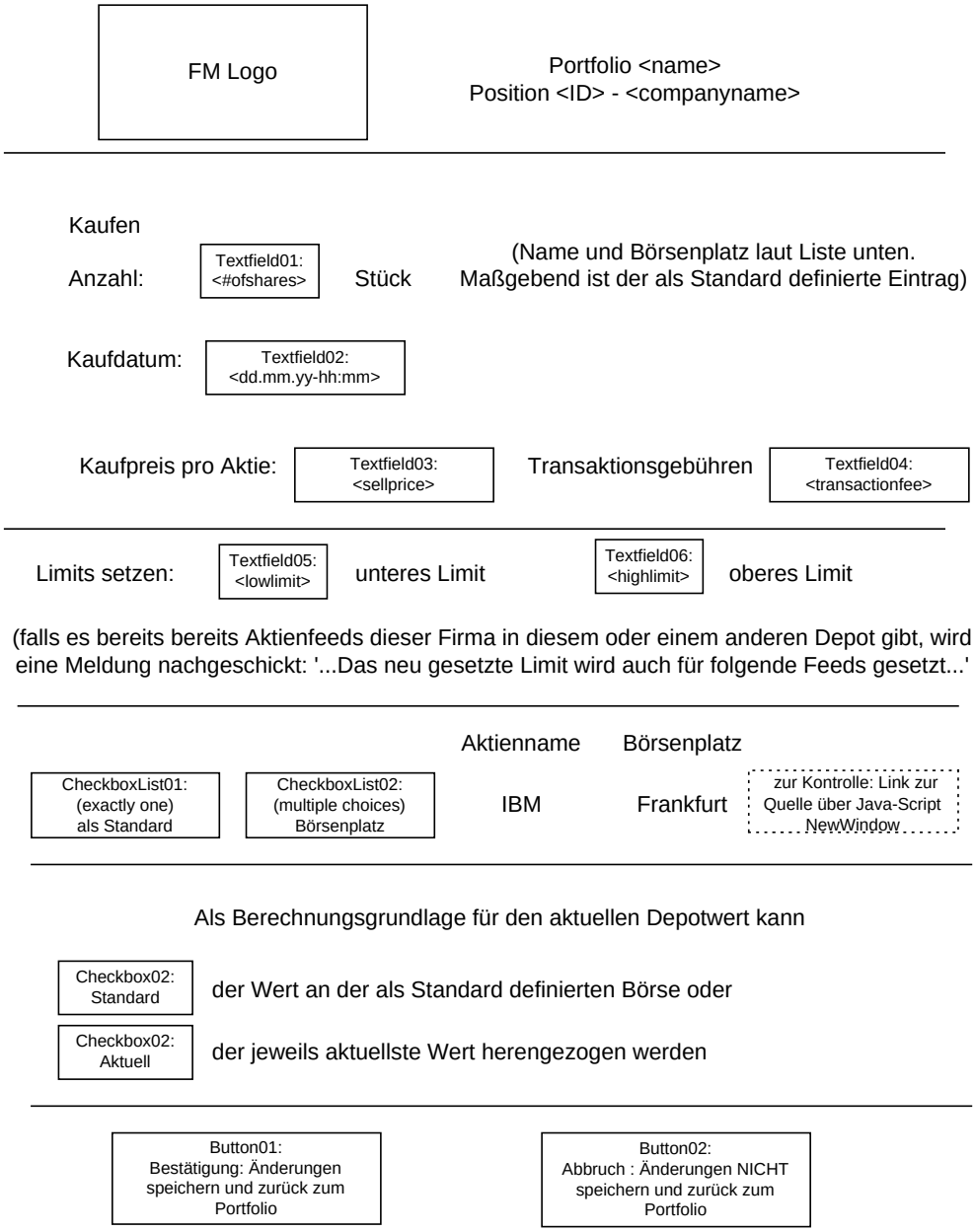



TA FMStocks change/delete position page

**TA FMStocks buy new shares page**

- displayed after ‘buy new shares’ was called from portfolio page
- allows the user to specify the parameters for a share buying transaction
- functions:
  - enter parameters for buying shares
  - enter paramters for adjusting limits
  - display a list of available sources (feeds for the shares)
    - the user may select or de-select one or more sources in the list

  - exactly one source must be selected to act as the default source
  - attached to each source is an link to external pages at value provider sites to allow the user to verify if a source feeds exactly the values the user is interested in; these links are opened in new browser windows
- the user may specify the sources that should be used to compute the actual value of their shares; options are: ‘always use the default source’ and ‘always use the most up-to-date source’
- save changes and back to portfolio
- delete changes and back to portfolio



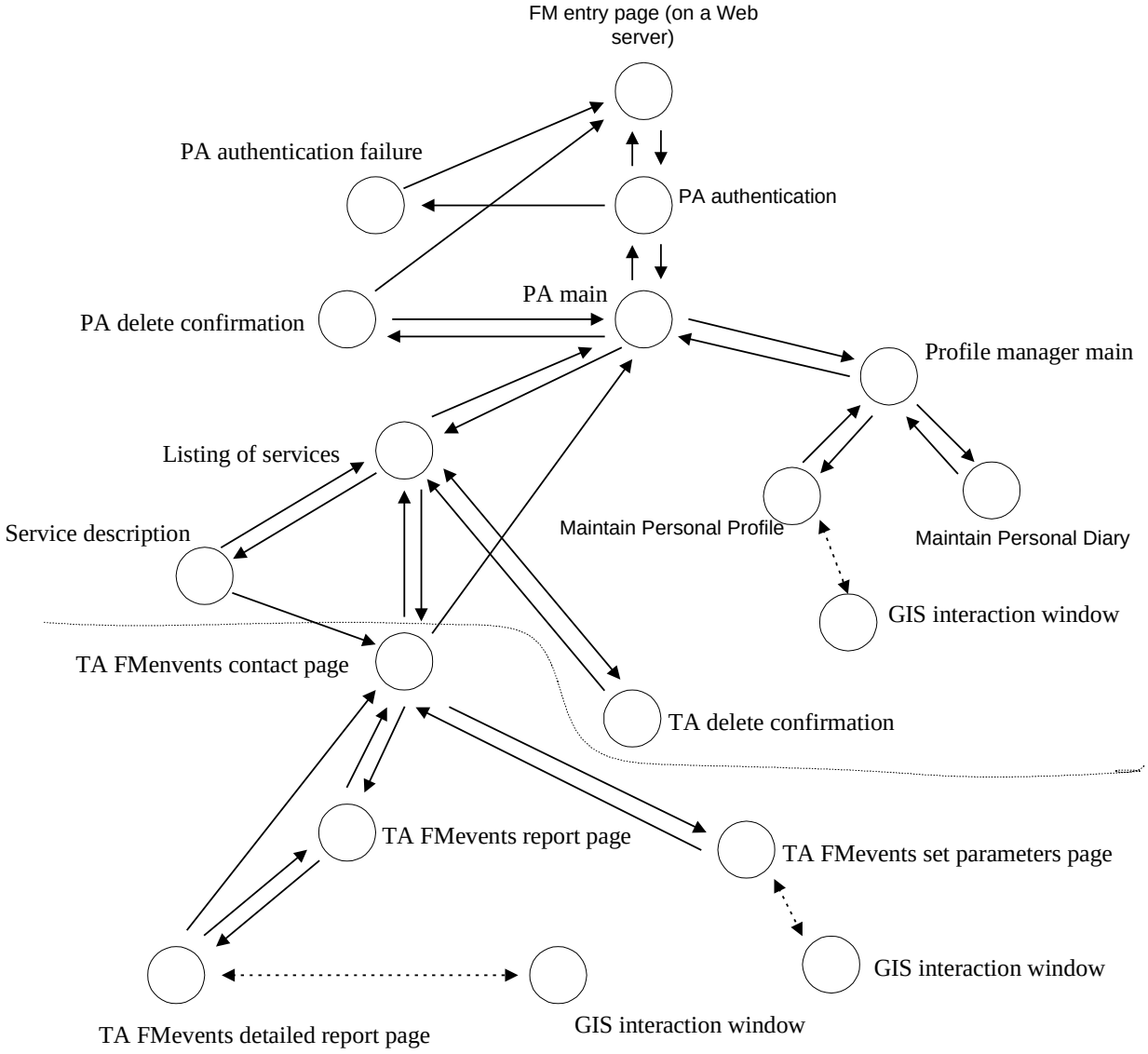

TA FMStocks buy new shares page

# 1.2 UI Navigation Diagrams

## 1.2.1 Navigational design for FMEventNotification

The following diagram illustrates the navigational design for the Event Notification System. The dotted line marks the borderline between PA and TA specific user interfaces.

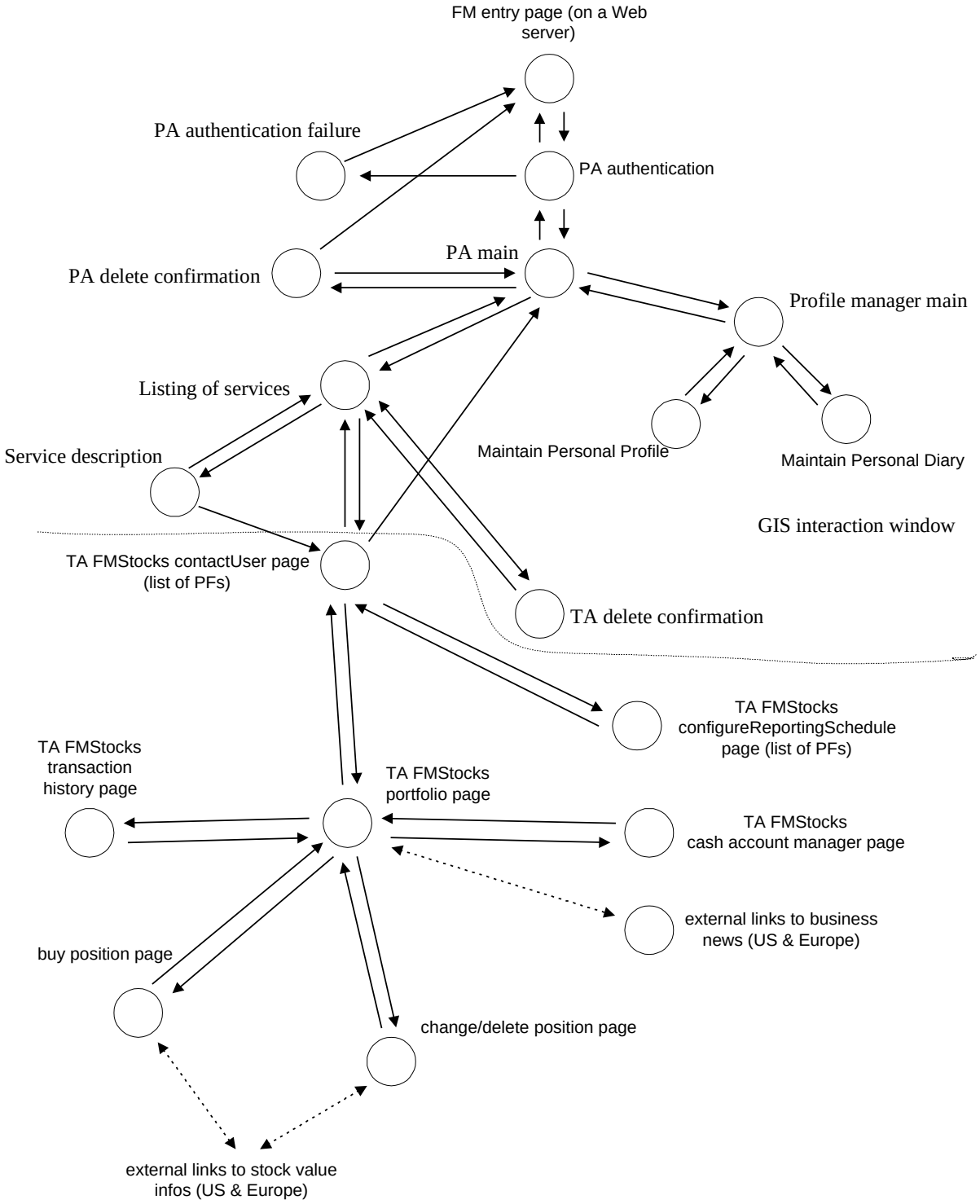
Note: Confirmation requests and pages for exception handling have been integrated only in a few instances (as examples). More of these pages might be required.




## 1.2.2 Navigational design for FMStockManager

The following diagram illustrates the navigational design for the Stock Management System. The dotted line marks the borderline between PA and TA specific user interfaces.

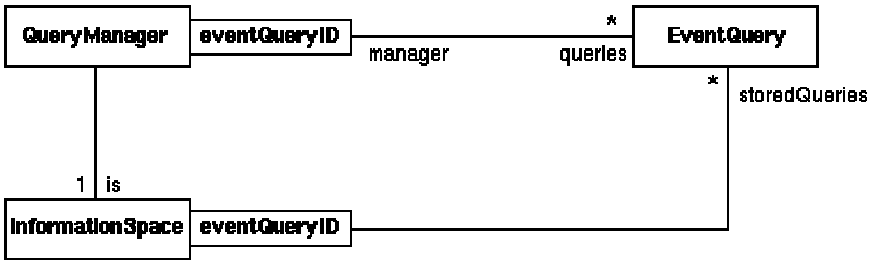
Note: Confirmation requests and pages for exception handling have been integrated only in a few instances (as examples). More of these pages might be required.

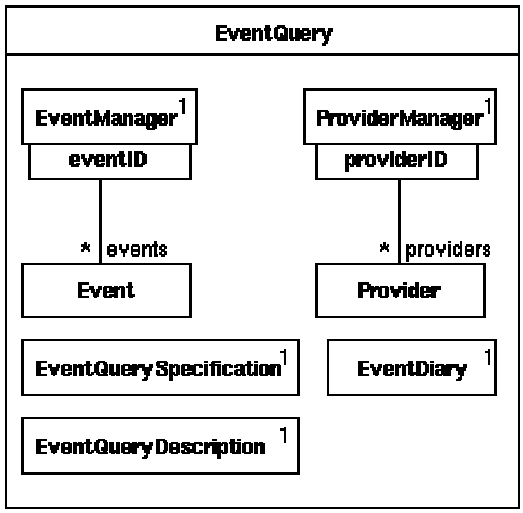



# *1.3 External application specific classes*

## 1.3.1 FMEventNotification specific classes

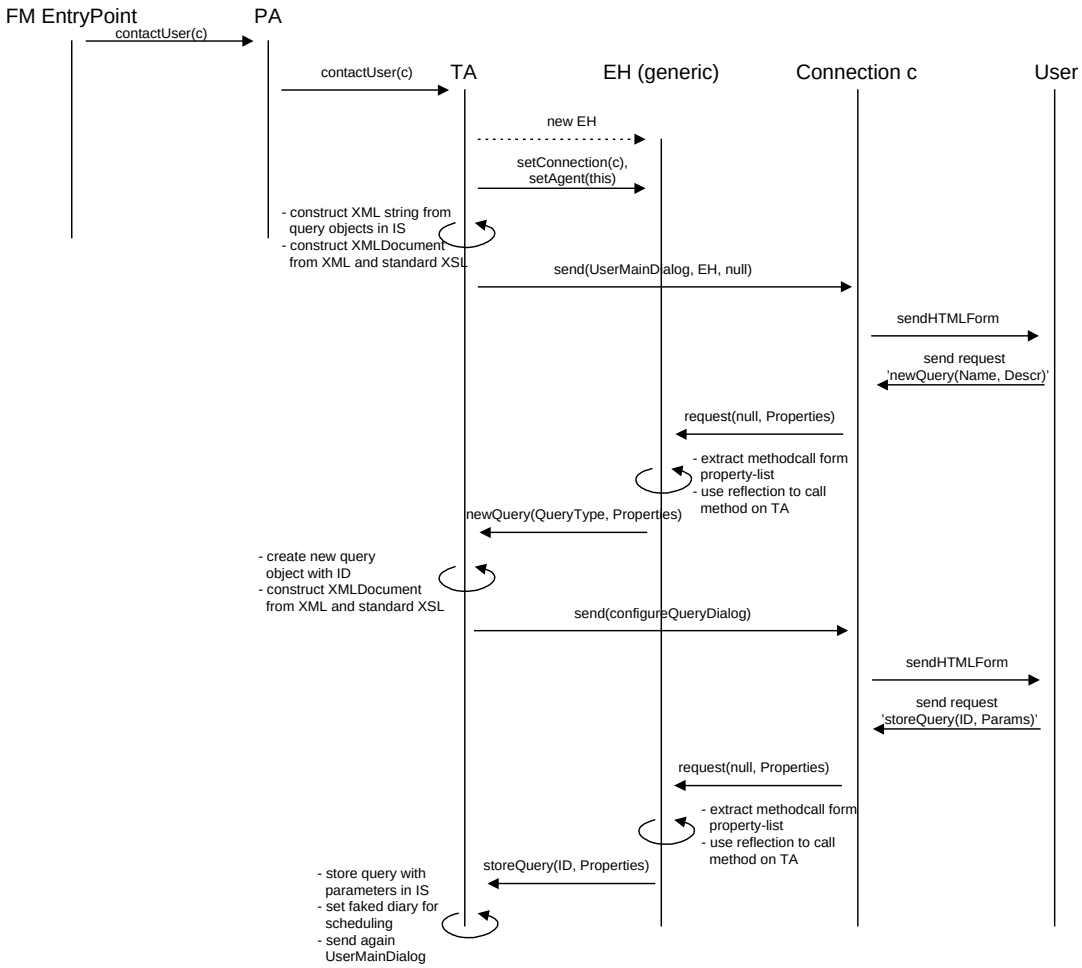
The following diagram illustrates the core classes used to implement the Event Notification System.

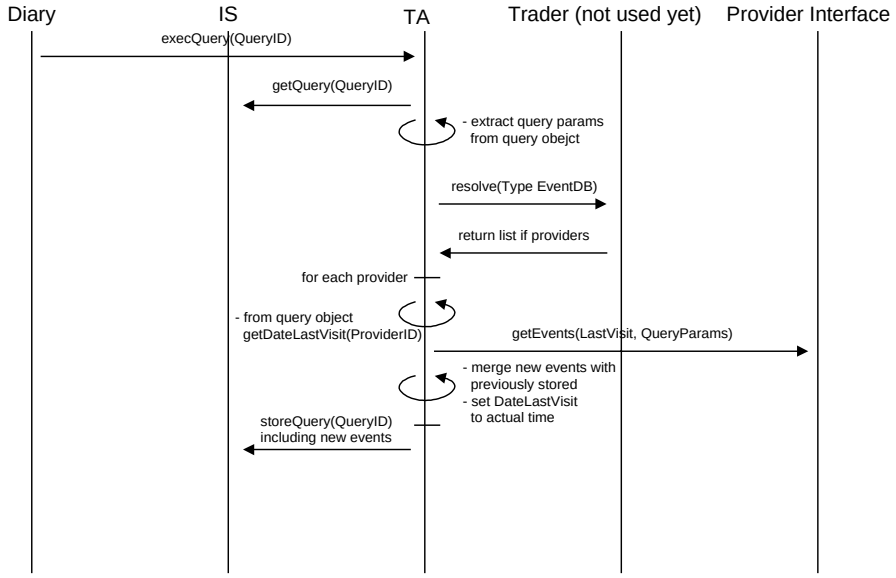


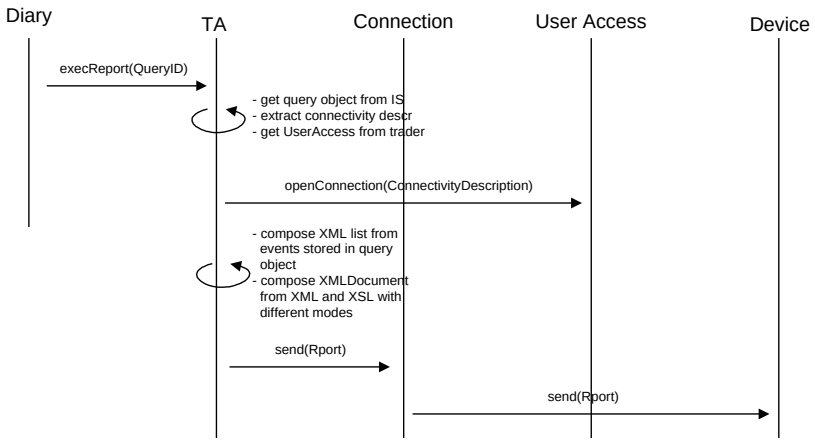




For a detailed description see the Java documentation for the Event Notification specific Java interfaces.

The next three diagrams show the control flow during the three major use cases within the Event Notification System: define a query, execute a query, deliver the results in form of a report.


FM EntryPoint
PA
TA
EH (generic)
Connection c
User
contactUser(c)
contactUser(c)
new EH
setConnection(c), setAgent(this)
- construct XML string from query objects in IS
- construct XMLDocument from XML and standard XSL
send(UserMainDialog, EH, null)
sendHTMLForm
send request 'newQuery(Name, Descr)'
request(null, Properties)
- extract methodcall form property-list
- use reflection to call method on TA
newQuery(QueryType, Properties)
- create new query object with ID
- construct XMLDocument from XML and standard XSL
send(configureQueryDialog)
sendHTMLForm
send request 'storeQuery(ID, Params)'
request(null, Properties)
- extract methodcall form property-list
- use reflection to call method on TA
storeQuery(ID, Properties)
- store query with parameters in IS
- set faked diary for scheduling
- send again UserMainDialog


Diary
IS
TA
Trader (not used yet)
Provider Interface
execQuery(QueryID)
getQuery(QueryID)
- extract query params from query obejct
resolve(Type EventDB)
return list if providers
for each provider
- from query object getDateLastVisit(ProviderID)
getEvents(LastVisit, QueryParams)
- merge new events with previously stored
- set DateLastVisit to actual time
storeQuery(QueryID) including new events

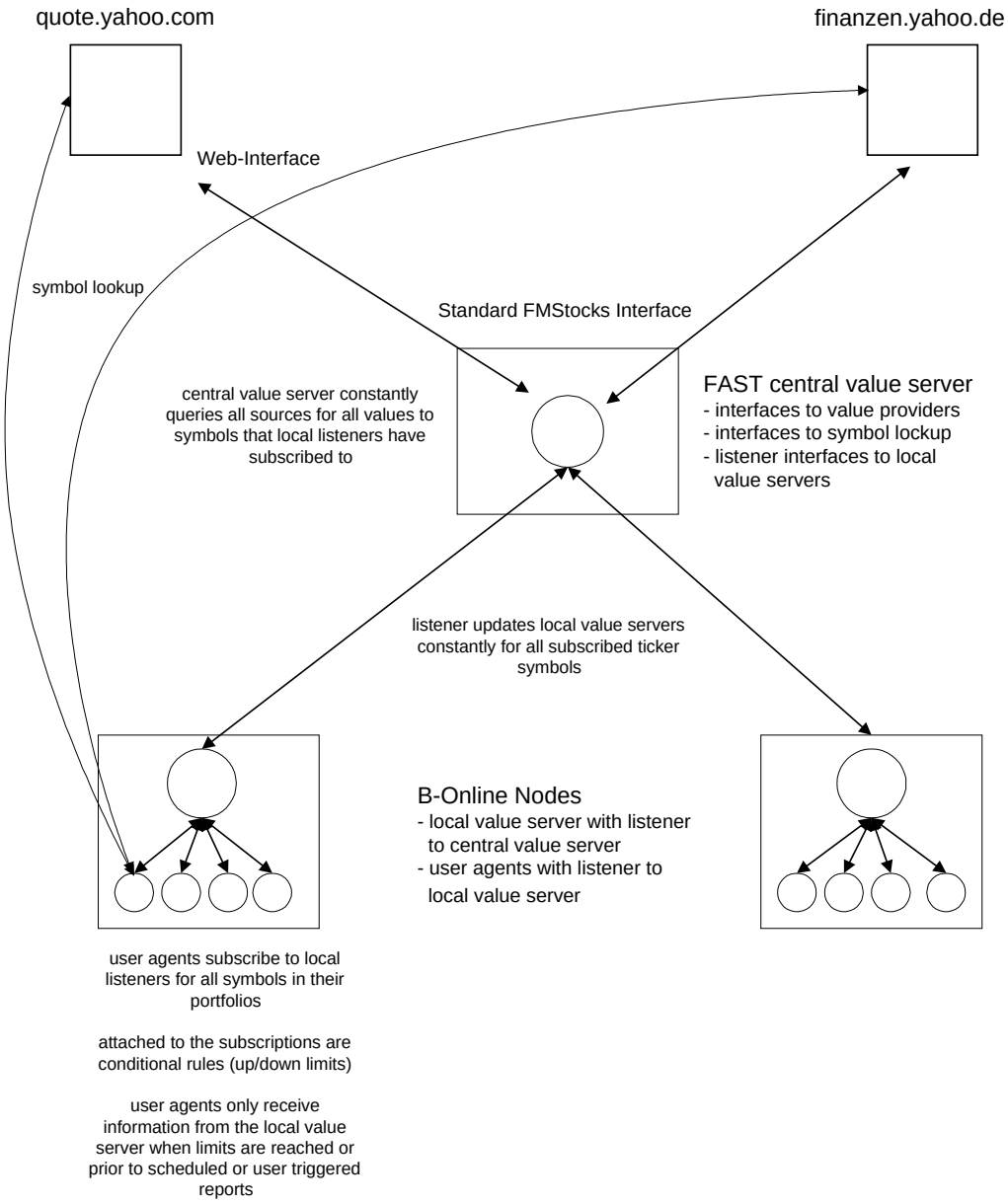



## 1.3.2 FMStockManager specific classes

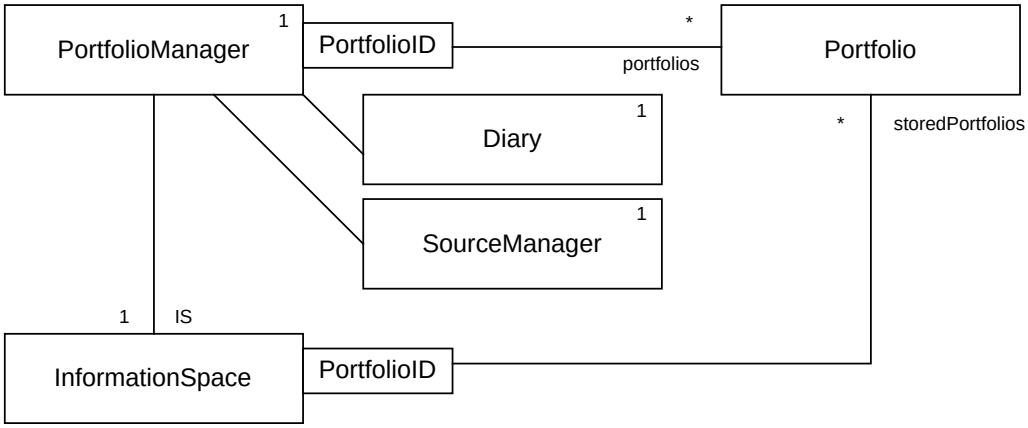
The Stock Portfolio Management System will consists of three application specific parts:

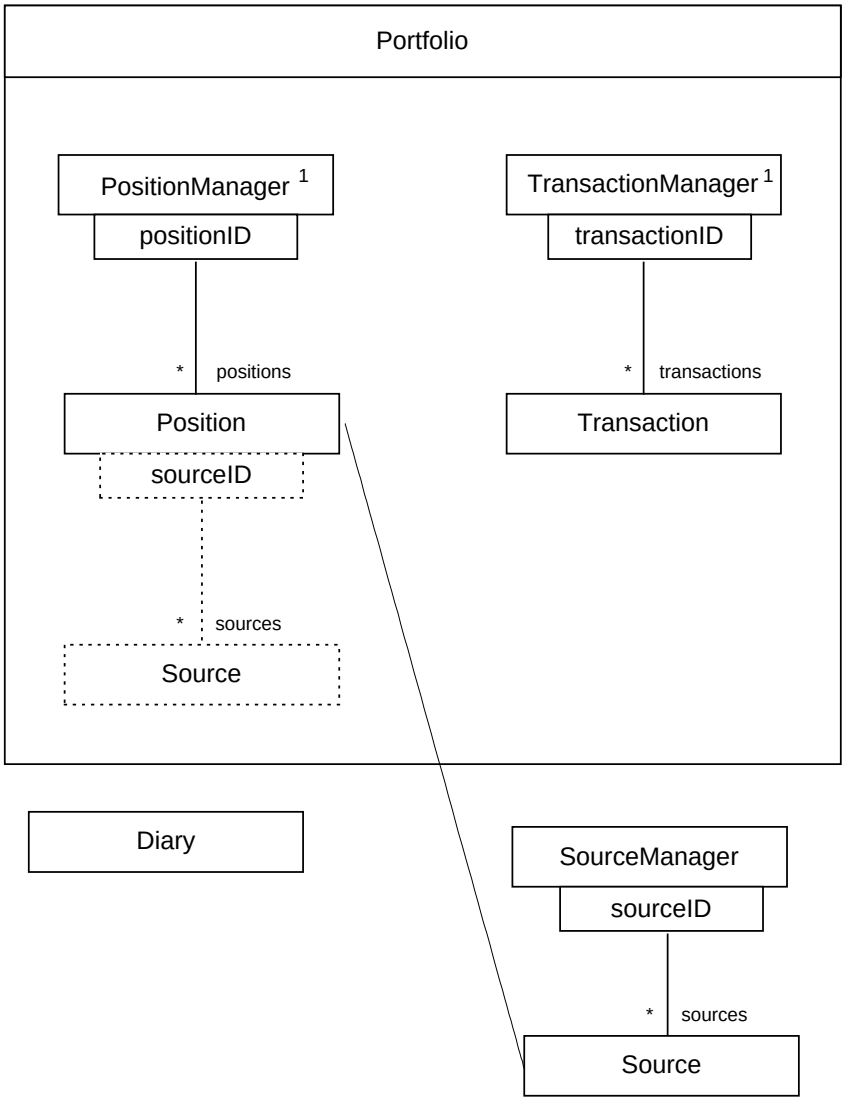
- the actual task agent (TA) managing a user's portfolios (one TA per user)
- a local value server (LVS) providing the share values to the TAs either on demand or triggered by limit settings (one LVS per host)
- a central value server (CVS) providing the LVSs with share values from various stock exchanges pulled from well known share value feeds on the internet (one CVS in the system)

The following diagram illustrates the roles of above described components:




The following diagrams illustrates the core classes used to implement the Stock Portfolio Management System.

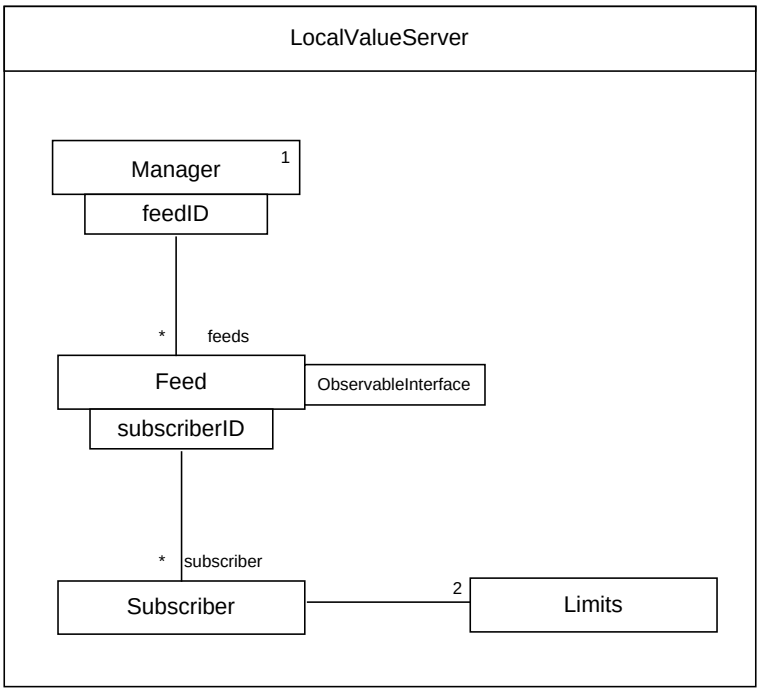


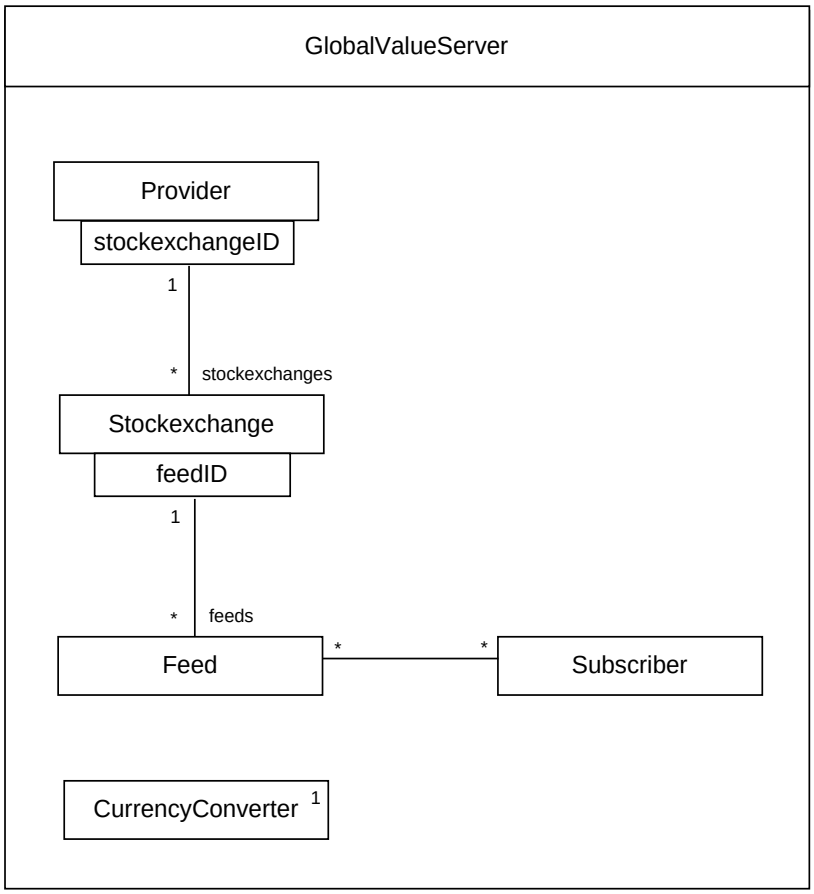




classes used to compose the TA

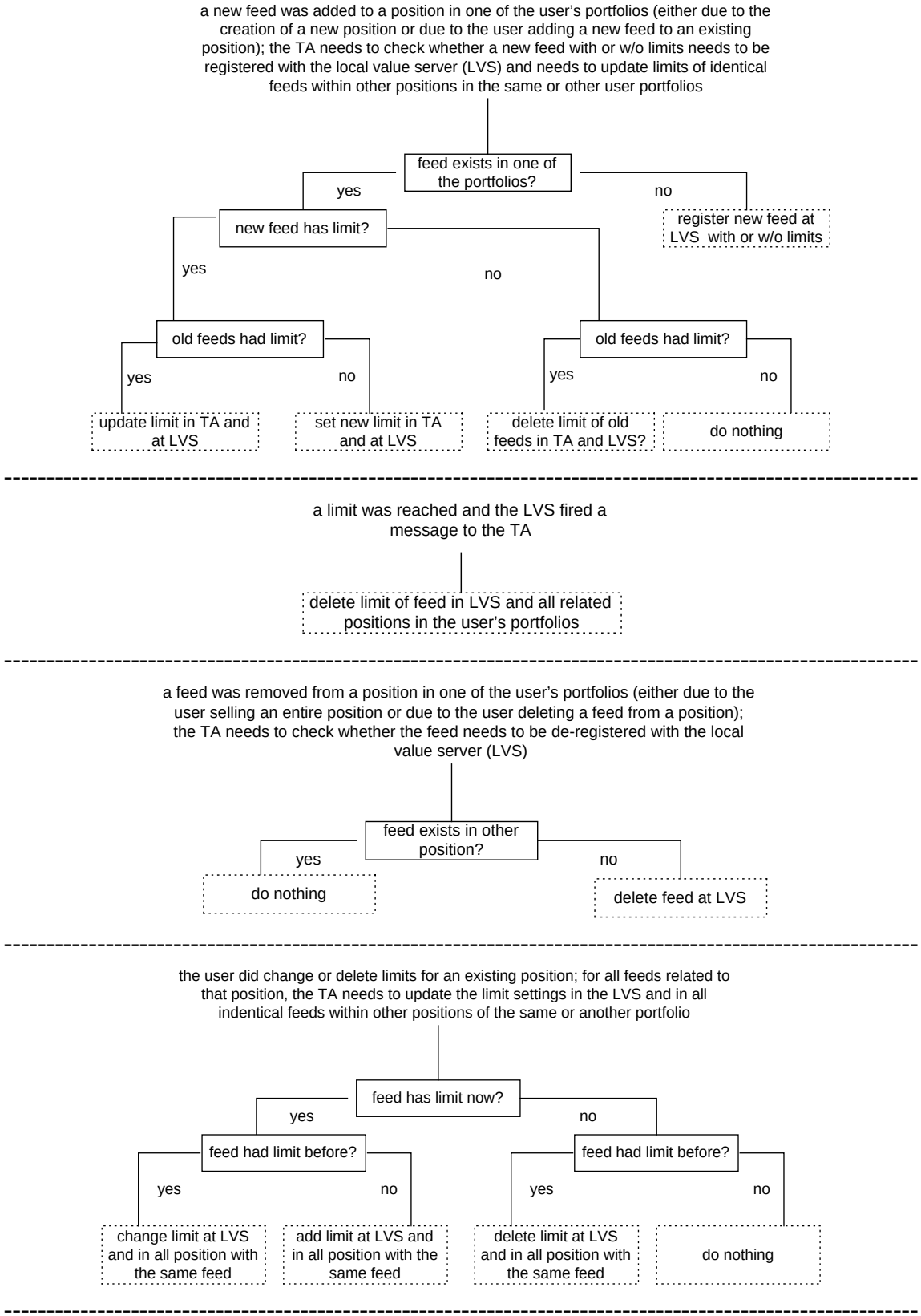


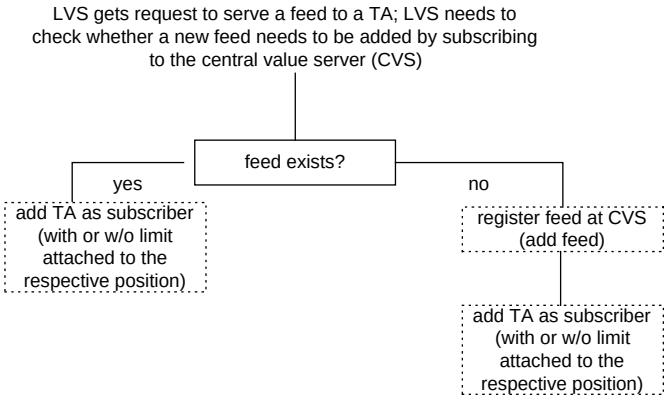

local value server class and sub-classes



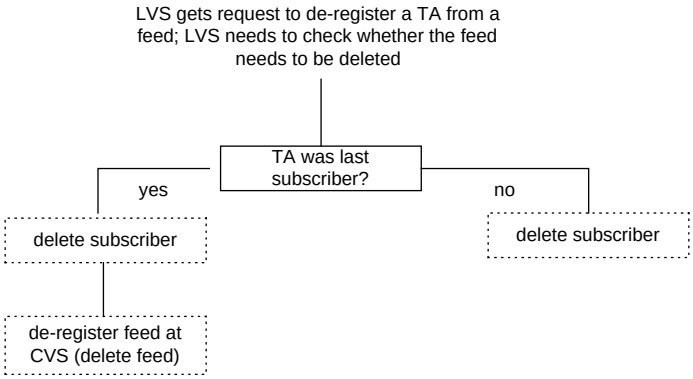

central value server class and sub-classes

The following diagrams show decision trees for (de-)registering value feeds between above described components (TA, LVS, CVS):

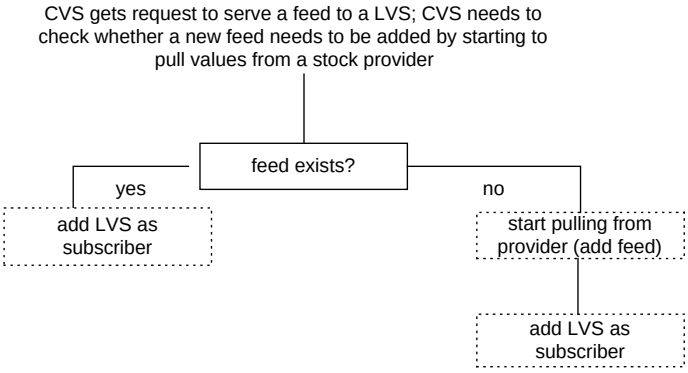



LVS gets request to serve a feed to a TA; LVS needs to check whether a new feed needs to be added by subscribing to the central value server (CVS)
feed exists?
yes
no
add TA as subscriber (with or w/o limit attached to the respective position)
register feed at CVS (add feed)
add TA as subscriber (with or w/o limit attached to the respective position)

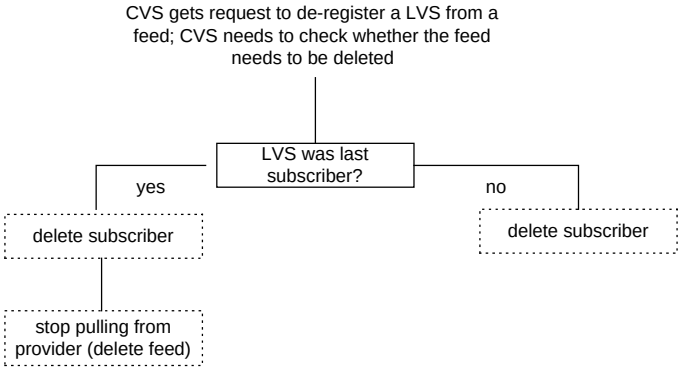
---


LVS gets request to de-register a TA from a feed; LVS needs to check whether the feed needs to be deleted
TA was last subscriber?
yes
no
delete subscriber
delete subscriber
de-register feed at CVS (delete feed)

---


CVS gets request to serve a feed to a LVS; CVS needs to check whether a new feed needs to be added by starting to pull values from a stock provider
feed exists?
yes
no
add LVS as subscriber
start pulling from provider (add feed)
add LVS as subscriber

---


CVS gets request to de-register a LVS from a feed; CVS needs to check whether the feed needs to be deleted
LVS was last subscriber?
yes
no
delete subscriber
delete subscriber
stop pulling from provider (delete feed)

# 1.4 Methods of the TA

**General note:**
All interaction with the user is implemented asynchronously. Preparation and delivery of user interfaces is strictly separated from handling the users feedback (i.e. for every user interaction there is a sendUserTheInterface method and a theUserRequestedThisAction method). That way the application cannot get stuck by waiting for user feedback.

## 1.4.1 Methods of the FMEventNotification TA

The methods described here form the main execution part of the TA. Once UWE's script-based agent framework will be available with integration of Information Space, Agent Diary, Personal Assistant and Personal Profile including the Personal Diary, these methods will be implemented as functions in a mission script.
The methods build on the applications core external classes specified in the previous section.

**Preconditions:**
The PA method contactUser(connection c) was called

**PA** {
// if required: create new TA with access to the user's IS;
// has a reference to a connection object 'c'
// knows about the FMEventNotificationTA interface 'contactUser'
FMEventNotificationTA.contactUser(connection c);
}

**Methods of TA:**

Upon constructing the FMEventNotificationTA an instance of FMEventNotification.FMQueryManager needs to be created and stored in the user's IS (named 'MyQueryManager'). The TA needs to have a handle to a section of the user's IS dedicated to the specific TA. All other application objects (e.g. new FMEventQueries) will be created using the FMQueryManager.
Furthermore the TA needs to have a handle to a diary object (named 'MyDiary'). Upon initializing, the TA should register at the diary by calling MyDiary.addDiaryListener(this).

**public void contactUser(connection c)** {
    eh = New FMEventHandling.FMEventHandler();
    eh.setConnection(c);
    eh.setAgent(this);
    userMainDialog();
}

**public void userMainDialog()** {

```
    String MyXMLList = MyQueryManager.getQueryListXML();
    // if this requires retrieving objects of type FMEventQuery via getFMEventQuery, all these need
    // to be freed via freeFMEventQuery after this operation!
    MyXMLListByteArray = New ByteArray(MyXMLList);
    UserMainDialogXMLDoc = New
        eh.FMXMLDocument.setExternalRepresentation(New
        Java.io.ByteArrayOutputStream(MyXMLListByteArray));
    UserMainDialogXMLDoc.setMimeType = 'XML';
    UserMainDialogXMLDoc.setXSL(loadURL(http://followme.fast.de/FMEventNotification/
        XSLs/userMainDialog.txt));
    // get XSL for the userMainDialog from a URL
    c.send(UserMainDialogXMLDoc, eh, null);
    // the last parameter is the QualityOfService and is not used at present
}

public void newQuery(QueryType, properties) {
        MyQuery = MyQueryManager.newEventQuery();
        // returns a new EventQuery object; NOTE: upon creating the new EventQuery the
        // QueryManager needs to pre-set the unique attributes QueryID, DeliverAlarmID and
        // QueryAlarmID; for this the QueryManager needs to hold index-lists or at least counters for
        // these attributes (maybe we need to do some string-integer conversion here, since QueryID
        // usually is just a string and therefore cannot be incremented!!
        // next set properties QueryName, Description and QueryID in the
        // EventQueryDescription
        from properties get:
                QueryName
                Description
        MyQueryDescription = MyQuery.getEventQueryDescription();
        MyQueryDescription.setName(QueryName);
        MyQueryDescription.setDescription(Description);
        QueryID = MyQuery.getEventQueryID;
        MyQueryDescription.setQueryID(QueryID);
        MyQueryDescription.setQueryType(QueryType);
        MyQueryManager.saveEventQuery(QueryID);
        // not that upon saving the new EventQuery object, the QueryManager must ensure that next
        // time an QueryManager.getQueryListXML is called, the new entry must be present -> if the
        // QueryListXML is saved separately for easier access (and not generated every time it is
        // requested by browsing through all EventQuery objects, calling .getQueryDescriptionXML
        // and composing them to the list), the QueryManager must update the list on every
        // EventQuery save operation!!
        MyQueryManager.freeEventQuery(ID);
```

```
        configureQueryDialog(QueryID);
}

public void configureQueryDialog(QueryID) {
        MyEventQuery = MyQueryManager.getEventQuery(QueryID);
        MyEventSpec = MyEventQuery.getEventSpecification
        MyEventSpecXML = MyEventSpec.getQuerySpecificationXML();
        MyEventDiary = MyEventQuery.getEventDiary();
        MyEventDiaryXML = MyEventDiary.getEventDiaryXML();
        MyQueryDescription = MyEventQuery.getQueryDescription(QueryID);
        // see above in the newQuery method: the in case of a new query, the EventQueryDescription
        // was already created but not saved in the IS
        MyQueryDescriptionXML = MyQueryDescription.getEventQueryDescriptionXML();
        MyXMLByteArray = New ByteArray(MyEventSpecXML + MyEventDiaryXML
        + MyQueryDescriptionXML);
        // the 3 XMLs need to be composed according to the specification of the XML (see below)
        ConfigureQueryDialogXMLDoc = New
        eh.FMXMLDocument.setExternalRepresentation(New
        Java.io.ByteArrayOutputStream(MyXMLByteArray));
        ConfigureQueryDialogXMLDoc.setMimeType = 'XML';
        ConfigureQueryDialogXMLDoc.setXSL(loadURL(http://followme.fast.de/
                FMEventNotification/XSLs/ ConfigureQueryDialog.txt));
        // get XSL for the userMainDialog from a URL
        c.send(ConfigureQueryDialogXMLDoc, eh, null);
        // the last parameter is the QualityOfService and is not used at present
        // the EventQuery and the EventQueryDescription need to be unlocked and kept in
        // memory
        MyQueryManager.saveEventQuery(QueryID);
        MyEventManager.freeEventQuery(QueryID);
}

public void configureQuery(QueryID, Properties) {
        MyEventQuery = MyQueryManager.getEventQuery(QueryID);
        MyEventSpec = MyEventQuery.getEventSpecification;
        from properties build:
                FMEventCategory
                FMEventLocation
                FMEventTime
                // includes the parameters MinTimeOffset and MaxTimeOffset
        MyEventSpec.setEventCategory(FMEventCategory);
        MyEventSpec.setEventLocation(FMEventLocation);
```

```
MyEventSpec.setEventTime(FMEventTime);
MyEventDiary = MyEventQuery.getEventDiary();
from properties build:
        DeviceToDeliver
        TimeIntervalFrom
        TimeIntervalTo
        TimeToDeliver
        WeekdaysToDeliver
        ReportingType
// ReportingType specifies whether only events should be included that have not been
// included in previous reports
MyEventDiary.setDeviceToDeliver(DeviceToDeliver);
MyEventDiary.setTimeIntervalFrom(TimeIntervalFrom);
MyEventDiary.setTimeIntervalTo(TimeIntervalTo);
MyEventDiary.setTimeToDeliver(TimeToDeliver);
MyEventDiary.setWeekdaysToDeliver(WeekdaysToDeliver);
MyEventDiary.setReportingType(ReportingType);
// next remove all IP entries
MyProviderManager = MyEventQuery.getProviderManager();
MyProviderManager.removeAllProviders();
// next remove all previously stored events (to be discussed!!)
MyEventManager = MyEventDiary.getEventmanager();
MyEventManager.removeAllEvents();
MyReportAlarm = MyEventDiary.getAlarmToDeliver();
// getAlarmToDeliver returns an interface of type AlarmInterface as specified in the
// Personal Profile design paper; the returned object contains all time dependent
// information but not the method to be attached to the alarm (this is set below).
MyQueryAlarm = MyEventDiary.getAlarmToQuery();
// the algorithm to derive the QueryAlarm from the ReportAlarm is encapsulated in
// the EventQueryDiary; for now querying is just one hour prior to reporting
// the diary must ensure that the new schedule is computed successfully (e.g.
// 0:30 on Monday minus one hour results in 23:30 on Sunday!!)
MyReportAlarm.set_attach(”execReport(QueryID, notOnline,
        DeviceToDeliver.Type, DeviceToDeliver.Number, DeviceToDeliver.Host)”);
MyQueryAlarm.set_attach(”execQuery(QueryID)”);
ReportAlarmID = MyEventDiary.getDeliverAlarmID();
MyDiary.deleteAlarm(ReportAlarmID);
QueryAlarmID = MyEventDiary.getQueryAlarmID();
MyDiary.deleteAlarm(QueryAlarmID);
// delete the old version of the alarm from the diary
MyDiary.set_alarm(MyReportAlarmm, ReportAlarmID);
```

```
        MyDiary.set_alarm(MyQueryAlarm, QueryAlarmID);
        //sets the new versions of alarms
        MyQueryManager.saveEventQuery(QueryID);
        MyQueryManager.freeEventQuery(ID);
        // What happens if an object is locked and the system crashed? Does it remain locked?
        // make sure, that the TA is never serialized and backed up within a state where an object is
        // marked as locked!!
        userMainDialog();
}

backToPA() {
        MyPA.contactUser(connection c);
}

destroy() {
        // calls the destructor of the TA
        // all objects related to the TA will be deleted from IS and memory
        // the TA informs the PA of its destruction
        // the TA ends
}

deleteQuery(QueryID) {
        MyEventManager.deleteQuery(QueryID);
        // Note: the QueryDescriptionListXML needs to be updated!!
}

execReport(QueryID, OnlineFlag, DeviceToDeliver.Type, DeviceToDeliver.Number,
                DeviceToDeliver.Host) {
        MyQueryManager.getEventQuery(QueryID);
        'get ReportingType from EventDiary';
        'if OnlineFlag = "online"' {
                'compute XML with event-list independent of ReportingType'
                // Java-Script in XSL enables client-side switching between ReportingTypes
                'send XML/XS to the still active connection c'
                // the switch is initially set to the value of reportingType
        }
        'else' {
                // in this case the execReport was called by the diary and DeviceToDeliver.Type
                // is not an HTML online connection; in this case a new connection needs be
                // established; there is no need for an event handler, since no answer from the
                // connection is required
```

```
                ‘compute XML with event-list depending on ReportingType’
                ‘if XML non-empty’ {
                        ‘if DeviceToDeliver.Type is ‘null’’ {
                                ‘get DeviceToDeliver from user personal profile diary entries’
                        }
                        MyConDescr = New ConnectivityDescription(DeviceToDeliver.Type,
                                DeviceToDeliver.Number, DeviceToDeliver.Host);
                        MyUserAccess = MyTrader.get(TheCorrectUA);
                        MyConnection = MyUserAccess.openConnection(MyConDescr);
                        ‘send XML/XSL to MyConnection’
                        // the event handler in the send command  can be set to null!!
                }
        }
        // free the locked EventQuery object!!
        MyQueryManager.freeEventQuery(QueryID);
}

getNewEvents(QueryID) {
        // called from interactive report page
        execQuery(QueryID);
        execReport(QueryID, OnlineFlag, DeviceToDeliver.Type, DeviceToDeliver.Number,
                DeviceToDeliver.Host);
}

execDetailedReport(QueryID, EventID, OnlineFlag, DeviceToDeliver.Type,
                DeviceToDeliver.Number) {
        // NOTE: switching from a view on all events to a more detailed view on only one event
        // could be done by using a sophisticated XSL (as long as all information required to build
        // the detailed description is already in the XML that was sent!)
        // this would make above method execDetailedReport obsolete and the data (XML) would
        // be sent only once – switching between views would take place on client side!!!

        // called from interactive report page
        MyQuery = MyQueryManager.getEventQuery(QueryID)
        MyEventManager = MyQuery.getEventManager()
        MyEvent = MyEventManager.getEvent(EventID);
        // new method!!
        MyEventXMLByteArray = new ByteArray(MyEvent.getEventXML());
        // new method!!
        MyEventXMLDoc = New
```

```
        eh.FMXMLDocument.setExternalRepresentation(New
        Java.io.ByteArrayOutputStream(MyEventXMLByteArray));
        MyEventXMLDoc.setMimeType = 'XML';
        MyEventXMLDoc.setXSL(loadURL(http://followme.fast.de/
        FMEventNotification/XSLs/ EventXSL.txt));
        c.send(MyEventXMLDoc, eh, null);
        MyQueryManager.saveEventQuery(QueryID);
        MyQueryManager.freeEventQuery(QueryID);
}

deleteEvent(QueryID, EventID) {
        // called from interactive report page
        MyQuery = MyQueryManager.getEventQuery(QueryID)
        MyEventManager = MyQuery.getEventManager()
        MyEventManager.removeEvent(EventID);
        MyQueryManager.saveEventQuery(QueryID);
        MyQueryManager.freeEventQuery(QueryID);
}

execQuery(QueryID) {
        MyEventQuery = MyQueryManager.getEventQuery(QueryID);
        MySpec = MyEventQuery.getEventQuerySpecification();
        EventCategory = MySpec.getEventCategory();
        EventLocation = MySpec.getEventLocation();
        EventTime = MySpec.getEventTime();
        IPList[] = Trader.resolve(type "EventDBs");
        // the trader interface is not yet specified!!
        // IPList is a list of interfaces to an IP; each interface must provide at least a method to
        // identify the IP (e.g. IPService.getIPID()) and a method to query the database
        // (e.g. IPService.getEvents(LastVisit, EventCategory, EventLocation, EventTime)
        MyProviderManager = MyEventQuery.getProviderManager();
        MyEventManager = MyEventQuery.getEventManager();
        for ('each IP in IPList') {
                ThisIPService.bindToServiceInterface();
                ThisIPID = ThisIPService.getIPID();
                MyProvider = MyProviderManager.getProvider(ThisIPID);
                // if there is no such entry (i.e. the provider is a new one), the ProviderManager
                // returns a new Provider with the respective ID and sets the DateLastVisited attribute
                // to 01.01.1900
                LastVisit = MyProvider.getDateLastVisited();
                EventList[] = service.getEvents(LastVisit, EventCategory,
```

```
            EventLocation, EventTime);
            ThisIPService.unbindFromServiceInterface();
            MyEventManager.merge(EventList);
            // this merges the newly retrieved EventList with the actual list of events
            // stored in the EventManager according to the following rules:
            // outdated events are deleted (this is the only time outdated events are cleaned up –
            // maybe there should be an extra cleanup method to check all stored EventQueries
            // for outdated events)
            // new events are added (including timestamp of update, timestamp of retrieval,
            // event_ID + provider_ID, reported-flag set to false)
            // existing copies are updated (new timestamp of update, new timestamp of retrieval,
            // reported-flag set to false)
            MyProvider.setDateLastVisited('now');
        }
        MyQueryManger.saveEventQuery(QueryID);
        MyQueryManger.freeEventQuery(QueryID);
}
```

## 1.4.2 Methods of the FMStockManager TA

**Preconditions:**
The PA method contactUser(connection c) was called

**PA** {
// if required: create new TA with access to the user's IS;
// has a reference to a connection object 'c'
// knows about the FMStockManagerTA interface 'contactUser'
FMStockManagerTA.contactUser(connection c);
}

**Methods of TA:**

**Note:**
Associated to a given position in a given portfolio are sources of the share values. However other positions in the same or in other portfolios could use the same sources.

Therefore sources are managed centrally be a source manager. The individual positions just add a pointer to one of the sources in the source manager to indicate that they use this source.

Sources in the source manager implement a Java observer that offer an interface 'update'. The source manager registers the sources with their update interfaces at the local value server which implements an observable where observers can register for updates.

**void contactUser(connection c)** {
    eh = New FMEventHandling.FMEventHandler();
    eh.setConnection(c);
    eh.setAgent(this);
    userMainDialog();
}

**void userMainDialog()** {
// displays a list of existing portfolios, an option to create a new portfolio and an option to change
// reporting paramters

}

**void newPortfolio(name, decription)** {
// creates an empty new portfolio with name and description
MyPortfolioManager.newPortfolio(name, description);
}

**void configureReportingDialog()** {
// retrieves the actual settings of the reporting schedule from the FMStocksApplicationManager
// and sends an XML-XSL to the connection so the user may adjust the parameters for reporting
// to their individual needs
}

**void configureReporting(properties)** {
// called when the user submits the form sent by configureReportingDialog
// stores the parameters in the FMStocksApplicationManager.Diary object
}

**void destroy()** {
// deletes the entire stock management application
// prior to self-destruction the TA needs to de-register from all feeds and delete all objects
// stored in the information space related to this TA
}

**void portfolioMainDialog(portfolioID)** {
// prepares an XML-XSL dialog to display the content of a specific portfolio
// uses ‘getValues’ to receive the latest values of portfolio positions
}

**void viewTransactionHistory (portfolioID)** {

// prepares an XML-XSL document to display the history of transactions for a specific portfolio
// different views can be filtered from the entries in the XML by client-side JavaScript embedded
// in the XSL
}

**void manageCashAccountDialog(portfolioID)** {
// sends an XML-XSL dialog displaying the actual amount of money in the cash account
// the user may enter a new transaction by specifying value, type and description
}

**void updateCashAccount(portfoiloID, properties)** {
// updates the object FMPortfolio.CashAccount of a specific portfolio
// this might be due to the user feedback from above manageCashAccountDialog or due to the
// user buying or selling stocks (see below)
}

**void maintainPositionDialog(portfolioID, positonID)** {
// this form displays actual values of a specific share position in a specific portfolio;
// in the form a list of sources (feeds) available for that position is displayed; the list is composed
// of two parts: first, all currently subscribed feeds are displayed; second, all available feeds minus
// the list of already subscribed is displayed; the latter is derived via the lookupSymbol method
// using the name parameter of the position as search string
}

**void updatePosition(portfolioID, positionID, properties)** {
// this method updates a specific position according to the feedback provided by the user
// this includes updating cash account, transaction history and feeds and limits provided
// by the local value server
}

**void sendReport(deviceDescription)** {
// prepares a report on the contents of all portfolios and sends it to the appropriate device
}

**void newPositionDialog(portfolioID)** {
// allows the user to specify the parameters for a share buying transaction
// this includes the selection of one or more sources (feeds) for the position
}

**void buyShares(portfolioID, positionID, properties)** {
// pre: portfolio MyPF exists

properties are:

...

valueSource = (ProviderID, StockmarketID, TickerSymbol)

// e.g. (yahoo-us, NYSE/NASDAQ, IBM) or (yahoo-de, Frankfurt, 914987)

PositionID = MyPF.PositionManager.newPosition;

MyPosition = MyPF.PositionManager.getPosition(PositionID);

//set position parameters (warning = false)

MyPF.CashAccount.update('some values')

//update cash-account

TransactionID = MyPF.TransactionManager.newTransaction

MyTransaction = MyPF.TransactionManager.getTransaction(TransactionID)

MyTransaction.setValues('some transaction values')

MyPosition.newSource(yahoo-us, NYSE/NASDAQ, IBM, default)

MyPosition.newSource(yahoo-de, Frankfurt, 914987, non-default)

// there has to be exactly one default source at any given time!

// if the original default source is deleted a new one must be set!

LVSManager = MyLocalValueServer.getFeedManager;

// if no listener exist, a new one needs to be created!!

LVSManager.modifyFeed(yahoo-us, NYSE/NASDAQ, IBM, ID of MyPF, ID of the position to be modified)

// the local value server should handle this request as follows:

- if the feed (yahoo-us, NYSE/NASDAQ, IBM) doesn't exist, it needs to be added by creating a new feed object and subscribing to an appropriate feed from the global value server
- if no listener for the TA exists, a new one needs to be created and the TA is added as a subscriber object related to the ID of the listener
- if no limit with the given portfolio-ID and position-ID is attached to the subscriber, a new one needs to be added according to the specified parameters; if such an entry already exists, the values for the limits need to be changed
- the listener related to the TA needs to be adjusted to the new alarm

MyPortfolioManager.savePortfolio(MyPF);

MyPortfolioManager.freePortfolio(MyPF);

}

}

**Methods of the local value server (LVS):**

**Methods of the central value server (CVS):**

# 1.5 XML and XSL listings

## 1.5.1 XML and XSL for FMEventNotification

**EventQueryList XML (userMainDialog.xml):**

```
<FMEventQueryList>
    <FMEventQuery>
        <QueryName> Kino </QueryName>
        <Description> Kino am Wochenende in Landshut </Description>
        <QueryID> Query01 </QueryID>
    </FMEventQuery>
    <FMEventQuery>
        ...
    <FMEventQuery>
    ...
</FMEventQueryList>
```

**EventQueryList XSL (userMainDialog.xsl):**

```
    Button00: destroy();
    table:
    for all entries in main.xml {
        attach Buttons {
                Button0Xa ‘Anfrageparameter ändern’: configureQueryDialog(QueryID)
                Button0Xb ‘Bericht erstellen’: execReportByUser(QueryID)
                Button0Xc ‘Anfrage löschen’: deleteQuery(QueryID)
        }
    new query area {
        input type = text, name = QueryName
        input type = text, name = Description
        ButtonN1 ‘regelmäßige Anfrage’: newQuery(regular)
        ButtonN2 ‘einmalige Anfrage zu einem bestimmten Zeitpunkt’: newQuery(onetime)
        ButtonN3 ‘einmalige Anfrage sofort’: newQuery(onetimeImmediately)
    }
    back button
        ButtonB1 ‘Zurück zum Personal Assistant’: backToPA()
```

**QueryIDParams.xml (configureQueryDialog):**

```
<FMEventQueryParameterList>
```

```
// start content of EventQueryDescription
    <QueryName> Sprachkurse </QueryName>
    <QueryDescription> Sprachkurse in meiner Umgebung </QueryDescription>
    <QueryID> Query07 </QueryID>
    <QueryType> regular </QueryType>
    // is in {regular, onetime, onetimeImmediately}
// end content of EventQueryDescription
// start content of EventQuerySpecification
    <OfInterestFMEventCategoryList>
        <FMEventCategory>
            <Level01> knowledge </Level01>
            <Level02> courses </Level02>
            <Level03> courses on languages </Level03>
        </FMEventCategory>
        <FMEventCategory>
            <Level01> bla </Level01>
            <Level02> blabla </Level02>
            <Level03> blablabla </Level03>
        </FMEventCategory>
        ...
    </OfInterestFMEventCategoryList>
    <OfInterestFMEventLocation>
    // in future versions multiple location parameters could be set
        <BaseLocation>
            <xCoordinate> 20,5 </xCoordinate>
            <yCoordinate> 23,7 </yCoordinate>
        </BaseLocation>
        <Radius> 0,004 </Radius>
    </OfInterestFMEventLocation>
    <OfInterestFMEventDate>
        <CalendarInterval>
            <Startdate> 01091998 </Startdate>
            <Enddate> 31121998 </Enddate>
        </CalendarInterval>
        <WeekdayPattern> Mon,Tue,Fri </WeekdayPattern>
        <DaytimeInterval>
            <Starttime> 16:00 </Starttime>
            <Endtime> 18:30 </Endtime>
        </DaytimeInterval>
    </OfInterestFMEventDate>
    <MinTimeOffset> 7 </MinTimeOffset>
```

```
        // ignored if type in {onetime, onetimeImmediately}
        <MaxTimeOffset> 14 </MaxTimeOffset>
        // ignored if type in {onetime, onetimeImmediately}
    // end content of EventQuerySpecification
    // start content of EventQueryDiary
        <DeliveryTime>
        // entries depend on type of query!
                <CalendarInterval>
                        <Startdate> 01081998 </Startdate>
                        <Enddate> 30111998 </Enddate>
                </CalendarInterval>
                // Startdate = Endate if type in {onetime, onetimeImmediately}
                <WeekdayPattern> Wed,Sat </WeekdayPattern>
                // ignored if type in {onetime, onetimeImmediately}
                <Daytime> 19:00 </Daytime>
                // ignored if type = onetimeImmediately
        </DeliveryTime>
        <ConectivityDescription>
                <Type> fax </Type>
                <Number> +498992004718 </Number>
                <Host> fast.de </Host>
                // not required if type = onetimeImmediately (always current online connection!)
                // it is not yet clear how this will be presented to the user??!!
        </ConectivityDescription>
        <ReportingType>
                all
        </ReportingType>
        // is in {all, newOnly}
    // end content of EventQueryDiary
<FMEventQueryParameterList>
```

**QueryIDParams.xsl (configureQueryDialog):**

list all the parameters stored in the eventQuerySpecificationXMLDoc XML according to the layout described in the layout drawings; the layout depends on the QueryType parameter stored in the XML

// the list of event categories available for selection should be stored in a central location

// the XSL should retrieve this list ??

Button01 ‘submit’ {

configureQuery(hasProperties, QueryID, ParameterList);

}

Button02 ‘cancel’: userMainDialog(noProperties, QueryID);

### 1.5.2 XML and XSL for FMStockManager

To be defined.

# Appendix A: Interface descriptions

See the online Java doc for the most up-to-date version.