Preliminary Version

FESA's protocol to properly manage the life cycle of an operational FESA Class and the operational Front-end configuration.

Context overview:

The purpose of this document is to describe the mechanism ensuring that any kind of upgrade of an operational FESA class will not break the control system. This responsibility cannot be assumed only by FESA because, an overall view of all the dependencies with the surrounding components is required to handle properly FESA Class upgrades, in order to know what can be done without breaking any system having a dependency with the FESA class requiring some upgrades. In this document, this external system managing the whole configuration of the control system is called "ConfigurationManager".At CERN, for instance, the component incarnated the role of the "ConfigurationManager" is the Control Configuration Data-base, because the CERN's Control system architecture is data-base centric in terms of configuration. In this note we will focus on all the different use-cases we have to support and for each of them we will describe the work-flow and the services expected from this "ConfigurationManager". A second very important issue addressed by this note, concerns the configuration of the operational Front-end. FESA3 introduces a new concept called FESA DeployUnit. A "DeployUnit" is a work unit used to configure operational front-ends.

Constraints:

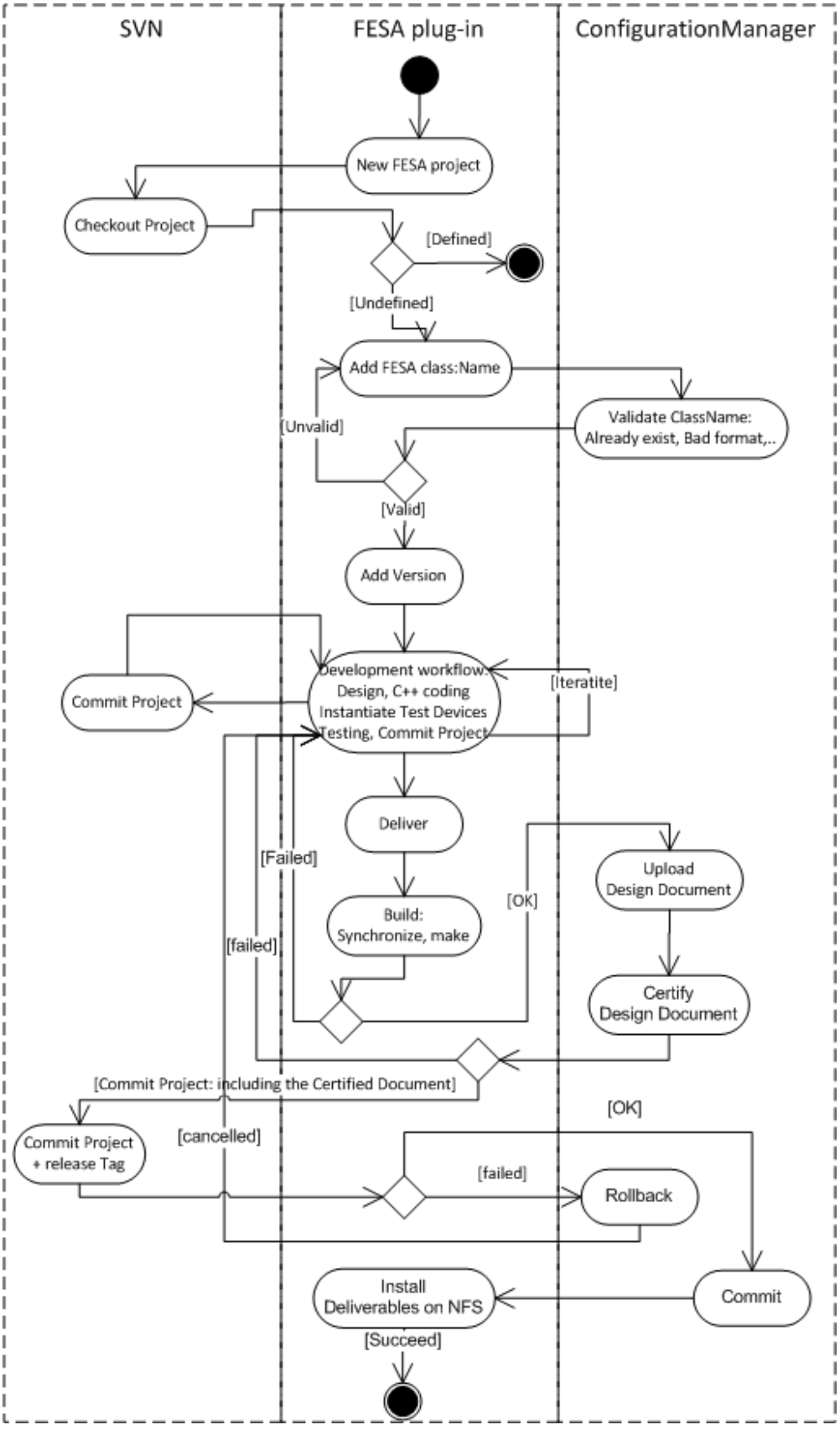
Even if the word "Data-Base" has been pronounced in the "context overview" chapter, it's crucial to remind that we want to provide an implementation which doesn't presume of the existence of any concrete implementation of the "ConfigurationManager", because seen from FESA world it's an external system. We want to formalize the problems in terms of concepts and let the implementation flexible and customizable per lab. Therefore, all the services expected from the "ConfigurationManager" should be defined by an abstract "Interface", that each lab has to implement.

FESA Class Work-flows:

FESA development tool is an Integrated Development Environment tool based on Eclipse and CDT (Eclipse C/C++ Development Tooling). This tool implements the complete work-flow required to develop, upgrade, and deliver a FESA class.

New FESA class development work-flow:

This workflow describes the complete process of developing a new FESA class



“ConfigurationManager” services:

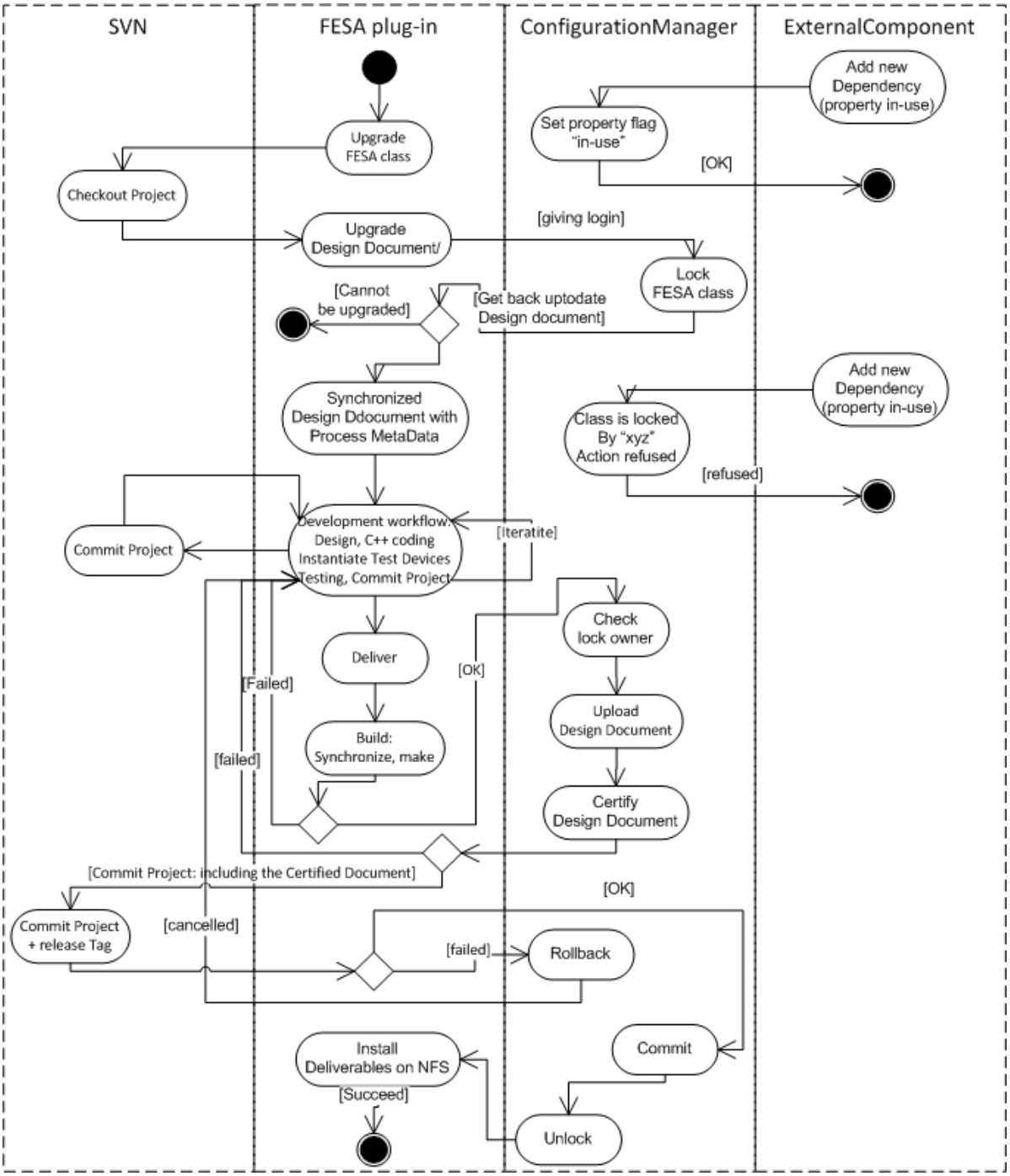
- “Validates Class Name”: defining a FESA Class Name, requires an overview of the current existing FESA classes to ensure the uniqueness of the ClassName, and also, to check the name against some “Naming Convention” for FESA class names. This service:
    - validates the new FESA class name, namely:
        - The class doesn’t exist already.
        - The name fulfills the constraints.
    - reserves the class Name: if one asks later for the same name the ConfigurationManager should not allocate it again, even if the FESA Class having this name has not been committed yet.
- “Upload Design Document”: delivering a FESA class for the first time means, to expose it to the operational world. For that reason, we have to obtain a kind of certification from the “ConfigurationManager”. This service has to be considered as the ultimate phase before committing the delivery. If this call succeeds, FESA plug-in gets back a certified document with a transaction ID.
    - CERN specific “Certification”: at CERN this phase consists to insert in the design document some “process metadata” which will be used to handle the future upgrade of the FESA class. Those process metadata are:
        - “operational”: this is a qualifier at the class level indicating that the FESA class is known in the operational world, in other words, the class can be deployed on operational front-ends.
        - “id”: basically the DataBase will assign a unique ID to each XML element. Thanks to this ID it will be possible to make the distinction between new elements or renamed elements while upgrading a FESA class
        - “in-use”: basically this flag will indicate an external dependency, making the element immutable. The only way of modifying/removing such an element will be first to cut the dependencies at the DataBase level.
        - “stamp”: stamping the document may be also useful.
- “Commit/Rollback”: using the transactionId returned by the previous call, the FESA plug-in will call “Commit” or “Rollback” depending if the remaining phases of the Delivery procedure succeed or not.
    - “Commit”: ultimate phase to commit the deliver for the “ConfigurationManager” .

- “Rollback”: leaves the “ConfigurationManager” as though the delivery procedure was never started.

- “install deliverables”: this is the ultimate phase of the delivery work-flow, consisting to deliver to a target place the deliverables, namely, the library and the header file.
  - At CERN: the target place is located on NFS

Upgrade Operational FESA class work-flow:

This workflow describes the process of upgrading an operational FESA class:



“ConfigurationManager” services: this second scenario representing the FESA class upgrade work flow, makes appear new services:

- “lock”: as soon as a class becomes operational, its set of properties, making the public API of the FESA class, are exposed and can be used by any external software component. Therefore the operational FESA class becomes like a “shared resource” that can be modified at the same time from two different places. On one side, the dependencies may evolve and this is managed by the “ConfigurationManager”, and on the other side, the developer may want to upgrade the FESA class. The proposal shown in the Activity diagram consists to require a “lock” from the “ConfigurationManager” giving the login for instance. From that moment, the FESA class definition is locked by the FESA plug-in, and any new request concerning a new dependency should be refused, knowing that if the request concerns a property which is already “in-use”, this one should be accepted. Thanks to this lock, the upgrade can be done in a safe way. This service requires an input parameter “login”, which can be very useful in case we have to notify bad clients letting the class locked for days. Furthermore, this call returns an up-to-date design document in terms of dependencies.

- “upload document”: previously describe, but in this scenario we can see the need of a kind of authentification in order to check if the one uploading the design is the one having the lock. So may be, while “uploading the document” we have to send the login which has been used to get a lock.

- “Upgrading rules”: the goal is to define the rules guaranteeing that the changes are backward compatible:

    - Interface:

        - property “in-use”: any operational property having this “in-use” attribute will be considered as immutable.

        - Expert properties or operational properties not in-use can be modified whitout constraints

    - Device-model:

        - fields

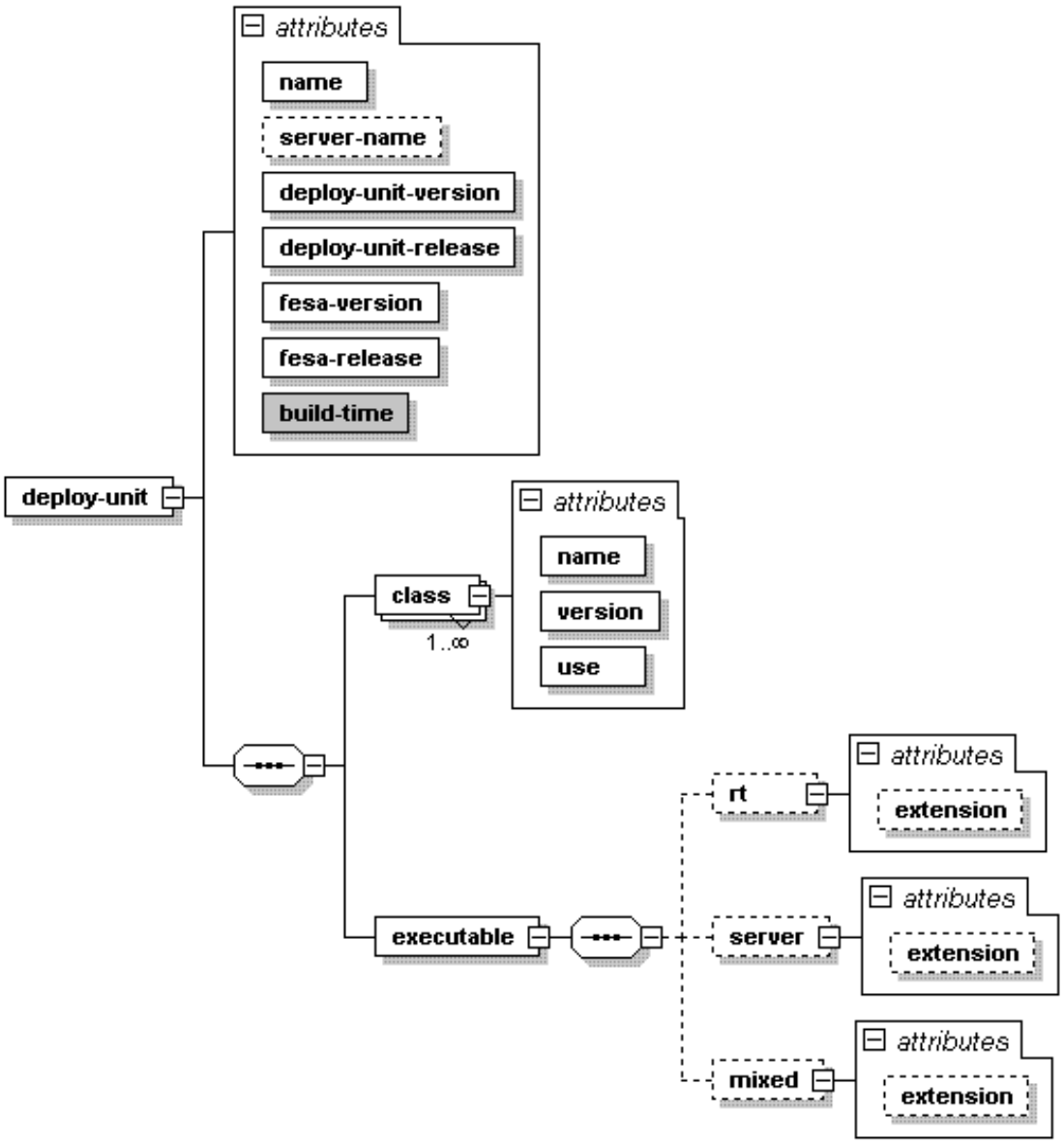
    - “id”: any element having

- “unlock”: consecutive to the “commit” phase, making the FESA class available to accept new dependencies. Probably we need an explicit call “unlock” in case where the developer decides to stop the upgrading process for instance….

Front-end Configuration

Front-end Configuration consist, among other things, to define the startup sequence, namely, the list of FESA executables to be started by a given Front-end at boot time. In order to achieve this goal, FESA 3 introduces the concept of “DeployUnit” which represents FESA executables and the rule to build them. Depending of the complexity of the accelerator device to be modeled, a “DeployUnit” can be made of one or many FESA classes linked together by means of relationships like inheritance, composition or association.

Deploy Unit Definition



- Deploy-unit attributes:

- name: gives the name of the deploy-unit. This name will be used to build the name of the executable (see executable). This name should be globally-unique, because it will be used to configure the FEC startup sequence. This uniqueness should be ensured by the ConfigurationManager.
- server-name: (NOT REQUIRED)gives the name of RDA name server. Again each instance of an equipment should have a globally-unique name and it should be the responsibility of the ConfigurationManager to define/assign it.
  - CERN: the current scheme to provide unique server-name consists to concatenate fesa-equipment name and host name separated by a point. For instance, "LTIM.pcgw18" is the server name for an instance of LTIM equipment deployed on the computer pcgw18.
  - FESA note: server-name is required in the TEST environment but not for operational Equipment document. Defining the attribute "optional" allows to use the same Equipment definition in both context, namely, "development" and "operation" .
- version: a deploy-unit has a version number. Typically, one delivered a new version of a FESA class involved by a "DeployUnit", and wants to deploy on a particular front-end a new version of the "DeployUnit" using the latest FESA class version.(verison should be appended to the executable name (see executable)
- release: gives the release number, which should be appended to the executable name (see executable).
- build-time: contains the time at which the executables have been created.

  NB(implementation note for FESA): to be sure to have the same "time" everywhere, we cannot rely on the gcc predefined macro (__TIME__) because it's the time at which the preprocessor is being run. So this "build-time" could be generated at the code generation.

- fesa-version(3-0, 3-1): the schema contains all the operational versions currently supported. Defining a "DeployUnitt" consists to select the appropriate fesa-version. Each "fesa-version-xy" element has two attributes:
  - version: fix value giving the fesa-version number.
  - release: one can select the appropriate fesa-release among all the releases of this particular fesa-version.
- class: defines all the classes embedded into the "DeployUnit". For each of them, some attributes are required:

- name/version : class name and version.
- use: one can select between “optional” or “required”. This attribute allows providing variable configuration of a single “DeployUnit”. For instance, considering a “DeployUnit” made of two FESA classes, A and B, and B, for instance, having the attribute “use” equals to “optional”. In this case , this “DeployUnit” could be deployed on a Front-end where the class B is not instantiated. Thanks to this flag, the Framework will not consider that the lack of the B instantiation document requires aborting the process.

- executable: specifies which kind of executables one wants to provide for this “DeployUnit”. Those different executables correspond to the deployment type supported by FESA, namely, “Split mode” leading to provide two separate executables, or “Unsplit mode” leading to provide a single executable. Each type of executable contains a single attribute “extension” having a fixed value, which will be appended to the final executable name.
  - rt : will build the real-time executable of the “equipment”.
    - extension attribute: fixed value equals to “_R”.
  - server: will build the server executable of the “equipment”.
    - extension attribute: fixed value equals to “_S”.
  - mixed: will be build an executable grouping the server and the real-time part of the “equipment”.
    - extension attribute: fixed value equals to “_M”

```
<deploy-unit name="ADeployUnit" version="1" release="2">
    <fesa-version-3-0 version="3.0" release="3.0.0">
            <class name="A" version="6" use="required" />
            <class name="B" version="0" use="required" />
            <class name="C" version="1" use="optional" />
            <executable>
                    <rt extension="_R"/>
                    <server extension="_S"/>
                    <mixed extension="_M"/>
            </executable>
    </fesa-version-3-0>
</deploy-unit>
```

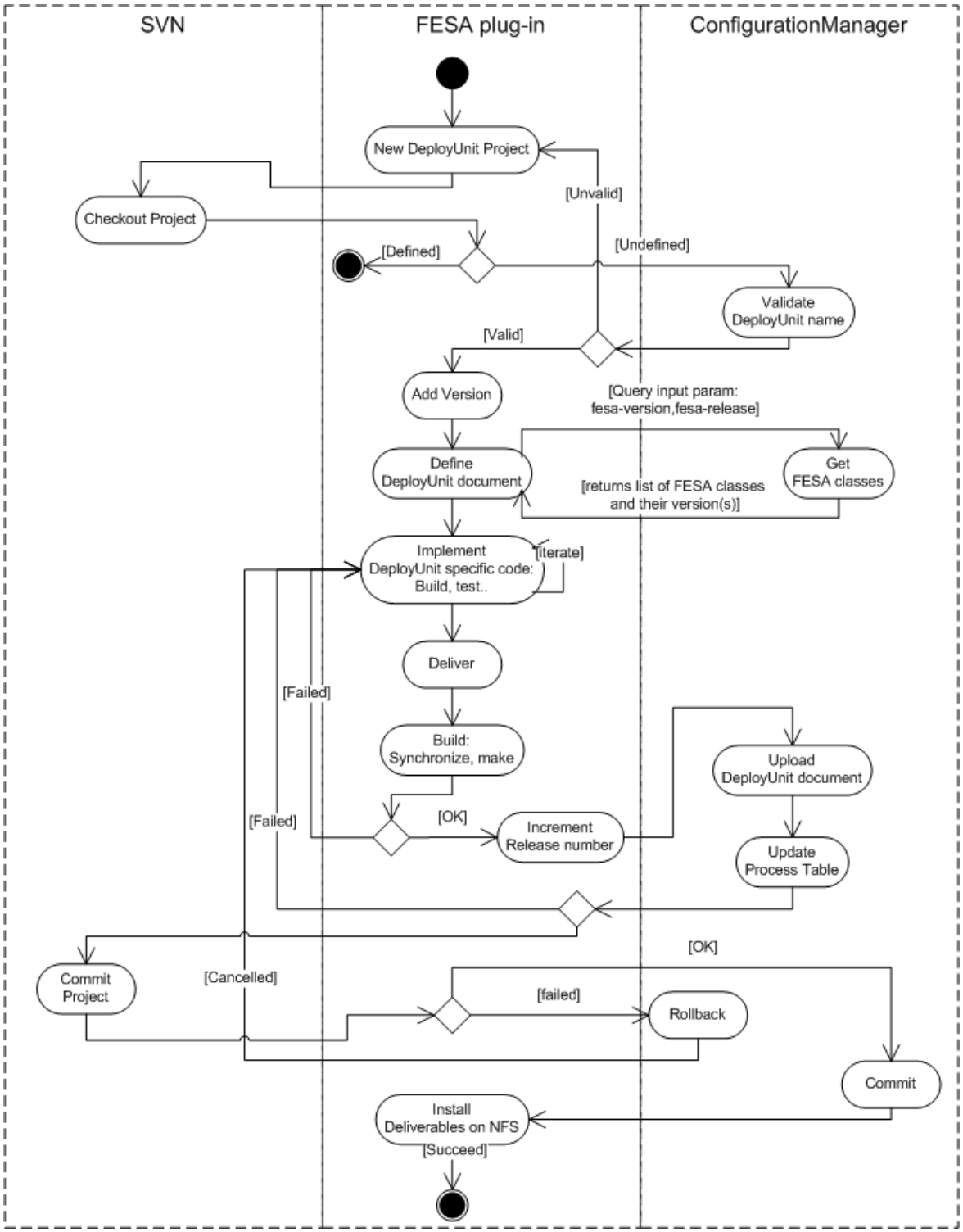
Assuming that “ADeployUnit” is globally unique, the algorithm to compute the final names of the executables is: “name + extension +version.release”, that is to say in this particular example:

- the real-time executable: ADeployUnit_R_1.2
- the server executable: ADeployUnit_S_1.2

- the mixed executable: ADeployUnit_M_1.2

New FESA DeployUnit workflow:

FESA plug-in defines a new type of project to manage the development of a “DeployUnit”. Basically, it follows the same steps as the FESA class development workflow, except than developing “Equipment” is obvious.



“ConfigurationManager” services:

- “Validates DeployUnit Name”: defining a “DeployUnit” Name, requires an overview of the current existing names to ensure the uniqueness of the “DeployUnitName”, and also, to check the name against some “Naming Convention”. This service:
    - validates the new “DeployUnit” name, namely:
        - Such name doesn’t exist already.
        - The name fulfills the constraints.
    - reserves the “DeployUnit” Name: if one asks later for the same name the ConfigurationManager should not allocate it again even if the “DeployUnit” having this name has not been committed yet.
- “Get Fesa Class List”: returns the list of FESA classes and their respective versions for a given “fesa-version” and “fesa-release”.
- “Upload DeployUnit Document”: upload the DeployUnit Design document into the ConfigurationManager.
- “Commit/Rollback”: using the transactionId returned by the previous call, the FESA plug-in will call “Commit” or “Rollback” depending if the remaining phases of the Delivery procedure succeed or not.
    - “Commit”: ultimate phase to commit the deliver for the “ConfigurationManager” .
    - “Rollback”: leaves the “ConfigurationManager” as though the delivery procedure was never started.
- “install deliverables”: this is the ultimate phase of the delivery work-flow, consisting to deliver to a target place the deliverables, namely, the library and the header file.
    - At CERN: the target place is located on NFS

FesaPlug-in requirement:

- “DeployUnit” will replace “Equipment”
- A “DeployUnit” project is attached to a single FESA-version. Therefore the FESA version should be selected when the “DeployUnit” project is created. In the “FESA Class Test” context, either the FESA-version is inherited from the “FESA class” or, the FESA-version has to be selected at the time “Add new DeployUnit” is pushed.
- DeployUnit Schema is FESA-version specific. Therefore this schema is not anymore located in the “Share” repository but in the FESA-version repository.

CERN ConfigurationManager behavior/expectations:

Current situation regarding FESA2:

- Double definition: currently Front-end configuration is done partially in FESA world (Front-end deployment document) and partially in the CCDB (Front-end startup sequence).
- No synchronization between FESA and CCDB: configuration can change in both sides without any synchronization mechanism. For instance, one can change the Deploy Document by removing a Fesa class or changing the deploy mode, but the startup sequence in the CCDB remains unchanged, and vice-versa, any change in the CCDB is not reflected in the FESA deploy document.
- We never know if what is running in the FEC is what we are browsing from the CCDB
- Potential "TimeBomb" while rebooting a process or the Front-end itself…

Conclusion, with FESA 3, it's time to give back to the CCDB the full role regarding the operational front-end configuration.

How FESA data will be used internally by the CCDB:

As described in the first part of this document, FESA provides the CCDB with two types of data:

- FESA class design document: each time a developer delivers a FESA class, the FESA class design document is uploaded into the CCDB.
- FESA DeployUnit document: each time a developer delivers a DeployUnit, the DeployUnit design document is uploaded into the CCDB.

Instantiating a FESA class:

Instantiating a FESA class consist to create device instances for a given FEC. In somehow, the developer wants to select a FESA class/version, select a Front-end and to create device instances. To achieve this goal, only the FESA class design document is needed.

Therefore, the outcome of this phase, is to have somewhere in the CCDB a list of device-instances for a given FESA class/version deployed on a given front-end.

Another very important responsibility of the CCDB, is to keep the device-instances synchronized with the FESA class design throughout the different upgrades that may occur. This issue is covered by the so-called "device Promotion", which promotes all device instances to make them compliant with the new FESA class design.

Link between "FESA class Design" and "DeployUnit"

A "DeployUnit" document contains all the dependencies with the FESA class involved to build the underlying executables. Assuming that the CCDB stamps all the FESA documents when they are

uploaded or committed, and thanks to the fact that each “DeployUnit” specifies its dependencies with respect to FESA classes, the CCDB should be able to know, what are the “DeployUnits” became obsolete following the FESA class changes.

Rule1: a “DeployUnit” document should have a stamp more recent than any stamps of the FESA class Design document involved in this” DeployUnit”. If it is not the case, the “DeployUnit” should be considered to be obsolete.

Extract Process names from “DeployUnit”:

From “DeployUnit” document the CCDB should fill in a kind of “process table”, from which one can pick-up the appropriate process to fill in the startup sequence of a Front-end. As described in the “DeployUnit Definition” chapter, a DeployUnit document contains all the information to know what are the underlying executables.

```
<deploy-unit name="ADeployUnit" version="1" release="2">
    <fesa-version-3-0 version="3.0" release="3.0.0">
            <class name="A" version="6" use="required" />
            <class name="B" version="0" use="required" />
            <class name="C" version="1" use="optional" />
            <executable>
                    <rt extension="_R"/>
                    <server extension="_S"/>
                    <mixed extension="_M"/>
            </executable>
    </fesa-version-3-0>
</deploy-unit>
```

Assuming that the executable names are build by appending “deploy_unit name”_”extension”_”version”.”release”, the CCDB receiving such a “DeployUnit” document fill in the “process table” with:

- ADeployUnit_R_1.2
- ADeployUnit_S_1.2
- ADeployUnit_M_1.2

Rule2: derived from the” Rule1”. Each time a “DeployUnit” document is uploaded, the CCDB extracts the underlying executables, updates the “process table” stamps each new inserted processes and updates and set the flag obsolete to false.

Rule3: Each time a “FESA Class Design” document is uploaded, the CCDB raises the “obsolete” flag for all processes having a dependency with the upgraded FESA class.

Configuring the Front-end Start-up sequence:

Probably as it is now, defining the start-up sequence of a front-end, consist to select from the "process list" the appropriate executable.

Rule4: From the process list, it should not be possible to select an obsolete process.

Rule 5: Each time a "FESA Class Design" document is uploaded, the CCDB knows what are the Front-end Start-up sequences became obsolete.

Extracting the Front-end Configuration

AT CERN, when a Front-end starts, he doesn't connect to the CCDB in order to get it's startup sequence, but it expects to find it from a proper location on the file system. This choice, that we don't want to change, requires a kind of offline extraction. Currently is done by running a script with two very important targets which are:

- transfer.ref: download the start-up sequence from the CCDB and install it on NFS
- new_dtab: download from the CCDB two FESA documents in order to re-generate the FEC executables:
    - FESA instantiation document.
    - FESA deployment document.

This workflow has many drawbacks:

- Binaries are rebuilt each time and furthermore for each Front-end even if the binary is shared by several front-end. The worst case is LTIM which is deployed on hundreds of front-end and generated hundreds of time.
- By rebuilding the executable, you never know if you don't bring a new version of one of the numerous libraries.
- Timebombs…
- Etc…

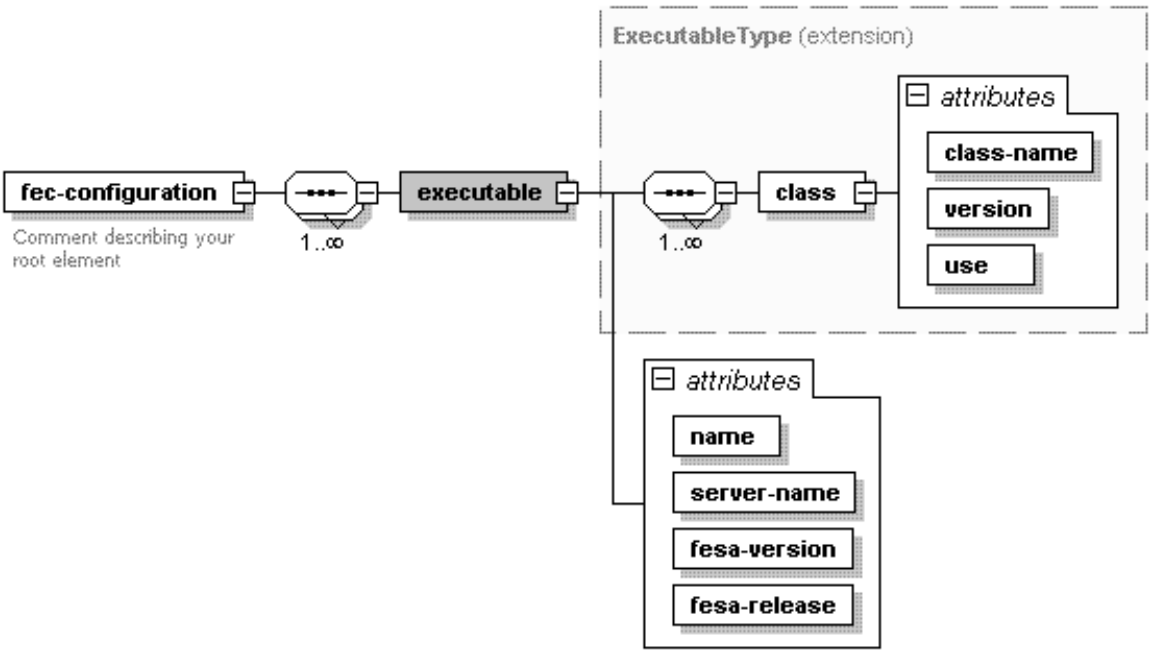
The new approach is supposed to face all those issues. We still have to extract from the CCDB the configuration data:

- transfer.ref: as now, download the start-up sequence from the CCDB and install it on NFS, but only if the Front-end start-up sequence is not obsolete.
  Rule6: the CCDB returns the transfer.ref sequence only if it is not obsolete.
- FesaConfig: download the FESA deployment document from the CCDB, but only if the Front-end start-up sequence is not obsolete.
  Rule7: the CCDB returns the FesaConfig document only if it is not obsolete.
- FesaInstantiate: from the FesaConfig document, the script extracts all the FESA class devices involved in the different executables and download successively the Fesa Instantiation documents giving the Fesa Class name/version

- NB for the script: the script should also set a series of symbolic link to the appropriate executable: /dsc/local/bin/ ADeployUnit_S_1.2 → /acc/dsc/[lhc,oper,lab]/CPU/fesaBin/classname/ ADeployUnit_S_1.2

Thanks to the rule 6 and 7, the "timebombs" issues are solved because it should not be possible to extract something from the CCDB which is obsolete.

Front-end Configuration document:



The Front-end Configuration document contains all the executables definition:

- Executable attributes
  - name: it should be the full name of the executable
  - server-name: is the Corba Name Server attributed by the CCDB to this executable. As discussed before , this name is unique and the CCDB is responsible to ensure the uniqueness of this name and also to assign to all devices instances served by this executable the same name.
  - fesa-version/fesa-release : identifies the fesa-version and the fesa-release used to build this executable.
- class: list of FESA classes contained in this executable. The attributes are:
  - class-name: name of the FESA class.
  - version: version of the FESA class.
  - use: attribute saying if the class is mandatory

Close Loop for Consistency Check:

The last point in the area of consistency which is not covered is to know precisely what is running in the Front-end. Of course this can be done manually by checking some config files or by comparing some binary dates with that start-up time of the process, but the objective is to automate this consistency check.

Proposal:

The FESA process contains all the information required to provide this consistency check, namely

- fesa-version, fesa-release
- deploy-unit release, build-time

Then, at the start-up time, when the FESA server connects to the CORBA name server to publish itself, it would be possible to expose all those internal parameters to make them visible and in particular, accessible by the Configurationmanager.

To achieve this goal, we have to negotiate with the MW team an extension of the current RDA interface. For the time being, when a FESA-server starts, it publishes only its name. The idea is to extend this API in order to transmit additional information.

Thanks to this mechanism, the loop is closed, and the ConfigurationManger should be in situation to detect if what is running on the operational Front-end is consistent with what is deployed.