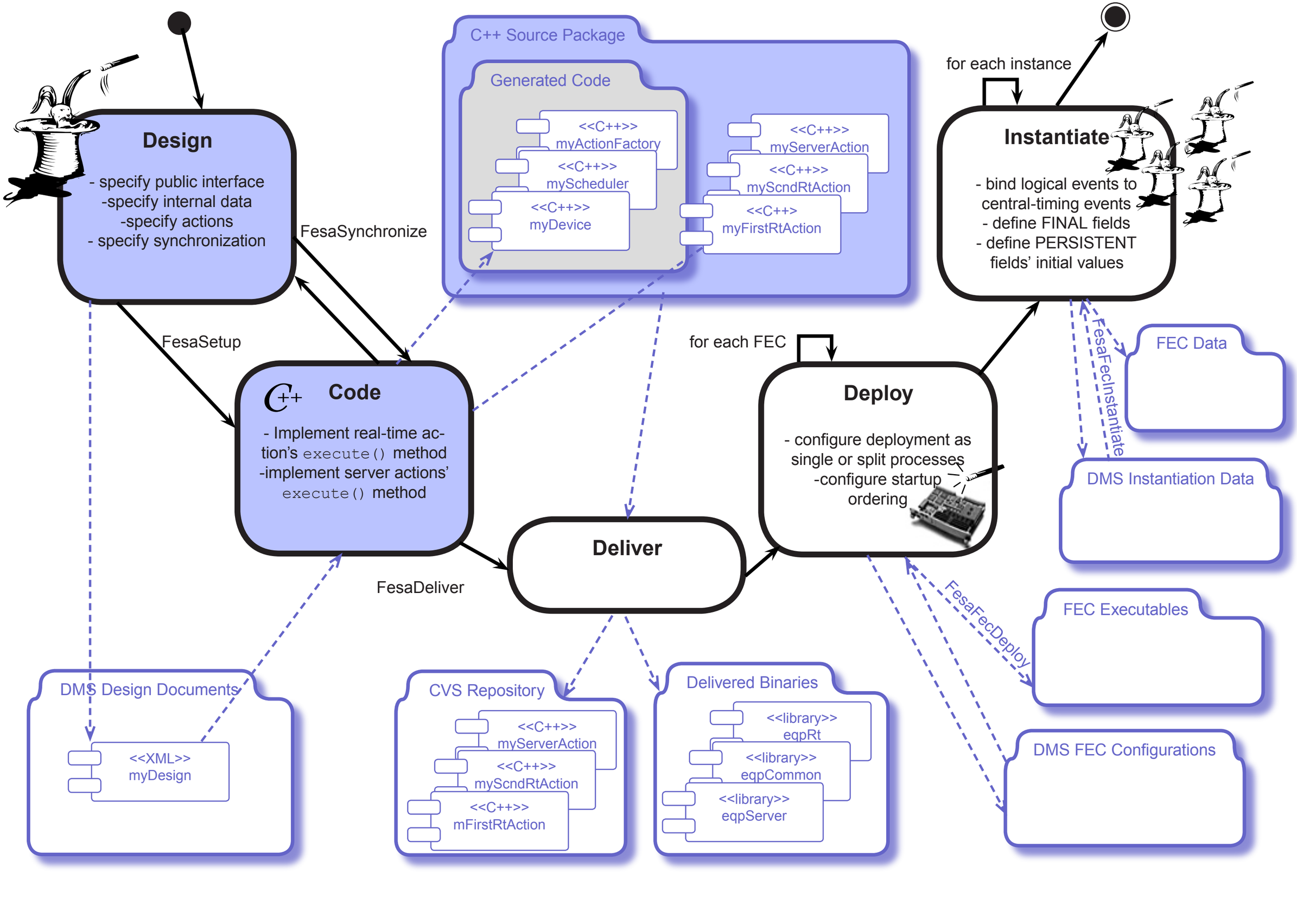
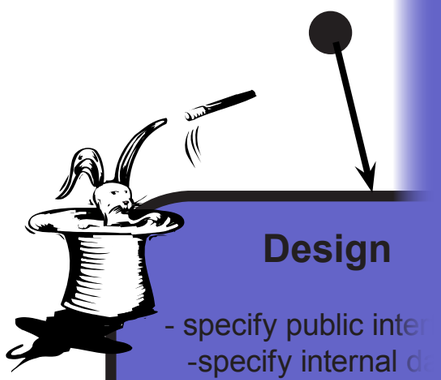

Design
- specify public interface
-specify internal data
-specify actions
- specify synchronization
FesaSynchronize
FesaSetup
C++ Source Package
Generated Code
<<C++>>
myActionFactory
<<C++>>
myScheduler
<<C++>>
myDevice
<<C++>>
myServerAction
<<C++>>
myScndRtAction
<<C++>>
myFirstRtAction
for each instance
Instantiate
- bind logical events to
central-timing events
- define FINAL fields
- define PERSISTENT
fields' initial values
FesaFecInstantiate
FEC Data
DMS Instantiation Data
for each FEC
Deploy
- configure deployment as
single or split processes
-configure startup
ordering
C++
Code
- Implement real-time ac-
tion's execute() method
-implement server actions'
execute() method
Deliver
FesaDeliver
FesaFecDeploy
FEC Executables
DMS FEC Configurations
DMS Design Documents
<<XML>>
myDesign
CVS Repository
<<C++>>
myServerAction
<<C++>>
myScndRtAction
<<C++>>
mFirstRtAction
Delivered Binaries
<<library>>
eqpRt
<<library>>
eqpCommon
<<library>>
eqpServer

Stage One: Designing your equipment's model



Design

- specify public inter
-specify internal d
-specify actions
- specify synchroniz

FesaSetu

1. Start the FESA design tool from the website developers' corner.

2. Click the button to start a new design.

3. Select the 'Trivial' template.

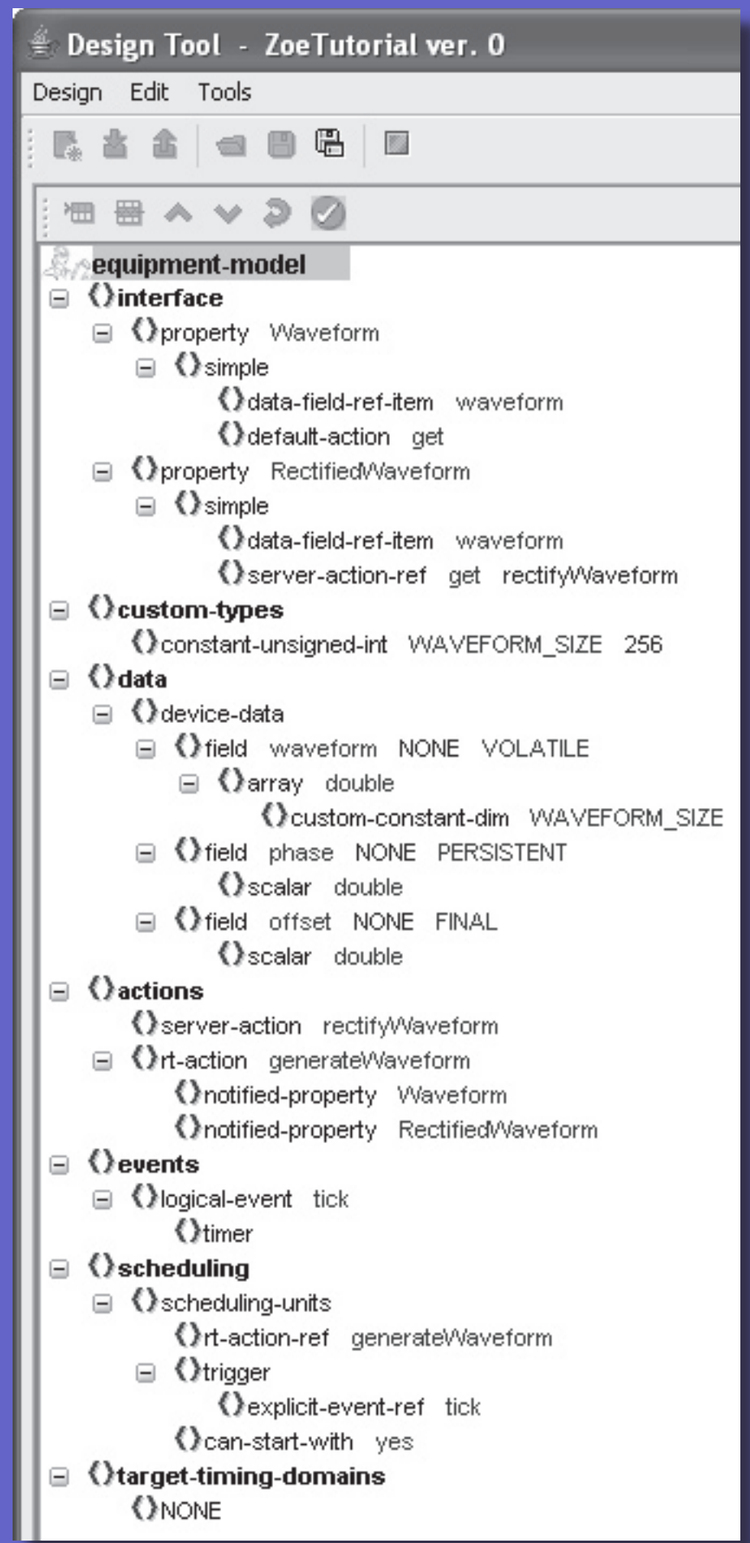
4. Create a design by adding elements as depicted on the screen-snaphot on the right. Devote sufficient time for you to understand the meaning of each element appearing in the model's tree.

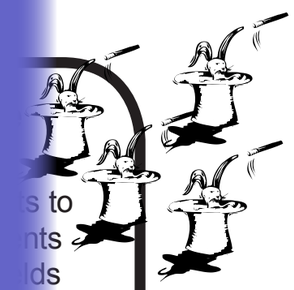
5. Make sure your design is well-formed by checking its validity with the button.

6. Press the button in order to save your design to the database.

7. Give your model a name and a version which clearly identify it as a fake equipment that you enter as part of a tutorial, e.g. `ZoeTutorial 0`

... from now on, your equipment is stored in the database. You will be able to bring subsequent changes to the model by restarting the design tool and retreiving it from the database with the button.





s to
nts
lds
ENT
es

FEC Data

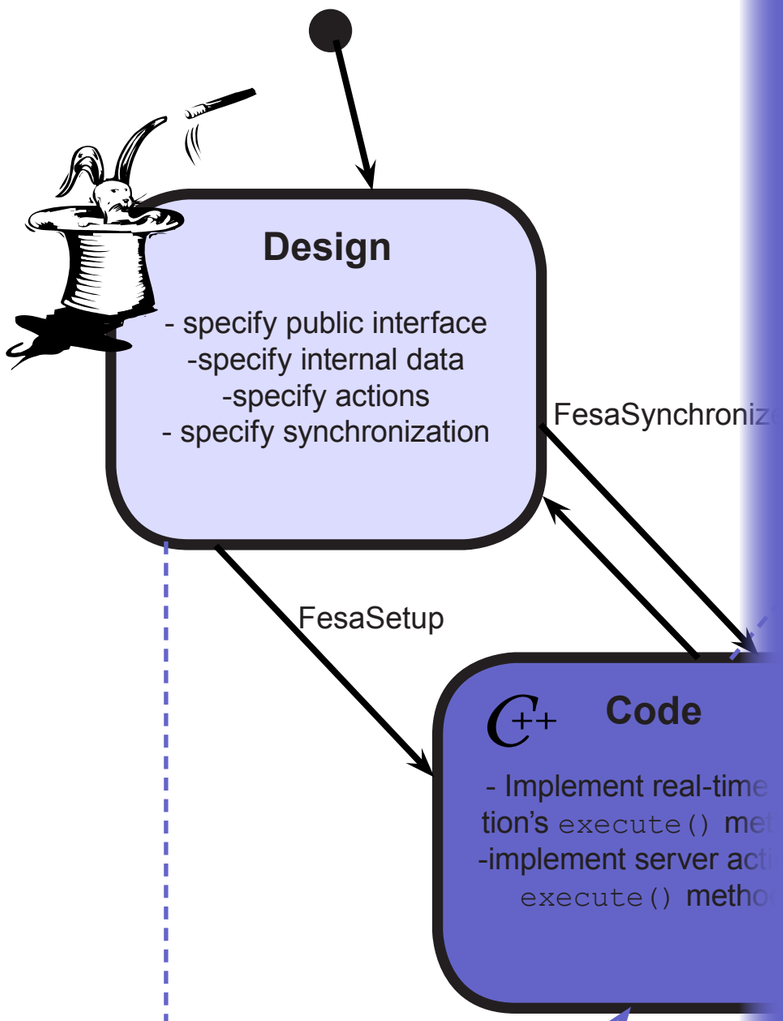
S Instantiation Data

xecutables

Configurations

DMS Design Document

<<XML>>
myDesign



Stage Two: Coding Server and RT Actions in C++

1. At the heart of your equipment's real-time activity, the `generateWaveform` real-time action class is meant (from your design) to be invoked each time a "tick" event occurs. You now need to enter the code of this action by filling-in the `generateWaveform::execute` method in the `generateWaveform.cpp` C++ implementation file.

```
void generateWaveform::execute(RTEvent* pEvent){
    for (unsigned int n=0;n<deviceCollection.size();n++){
        JoeTutorialDevice* pDevice=deviceCollection[n];
        double *pWaveform=pDevice->waveform.get();
        double phase=pDevice->phase.get();
        double offset=pDevice->offset.get();
        for (unsigned int i=0;i<WAVEFORM_SIZE; i++){
            pWaveform[i]=sin(2*3.14*i/100.0+phase)+offset;
        }
        pDevice->phase.set(phase+0.1);
        log<<"device has been woken-up by tick event "
            <<pDevice->name.get()
            <<" to generate its own waveform"
            <<endDebug;
    }
}
```

Preliminary: `Fesa Setup`

Before being able to enter your specific C++ code, you must first prepare a local work environment on Linux: invoke `Fesa Setup` for initial code-generation (or `Fesa Synchronize` for future design iterations).

```
Zoe> mkdir fesa
Zoe> cd fesa
Zoe/fesa> Fesa Setup ZoeTutorial 0
                              scratch
Zoe/fesa> cd ZoeTutorial/v0
Zoe/fesa/ZoeTutorial/v0> make
```

2. You don't need to provide any piece of code for accessing remotely the device to be able to retreive its waveform since the design specified a default `get` action to serve the `Waveform` property. On the other hand, must supply code that performs data shaping in order to implement the RectifiedWaveform property. To this end you implement the `rectifyWaveform.cpp` file in a fashion similar to the above. The `pWorkingDevice` automatically points to the device actually which is the target of the remote-access request.

```
void rectifyWaveform::execute(RequestEvent *pEvent){
    double *pWaveform=pWorkingDevice.waveform.get();
    for (int i=0;i<WAVEFORM_SIZE,i++) {
        value.rectifiedWaveform[i]=fabs(pWaveform[i]);
    }
    log<<pDevice->name.get()<<" remotely accessed"<<endDebug;
}
```

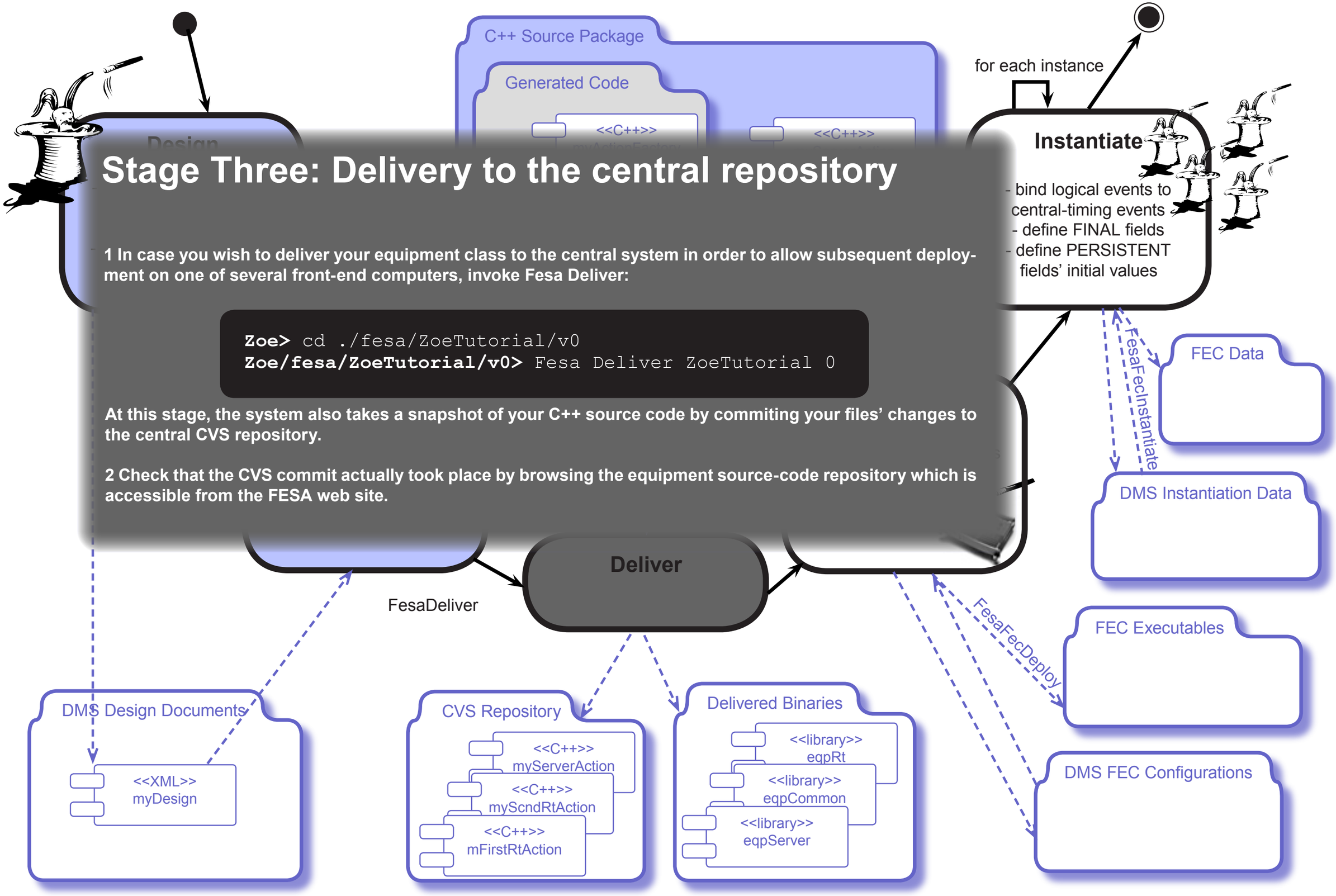
Stage Three: Delivery to the central repository

1 In case you wish to deliver your equipment class to the central system in order to allow subsequent deployment on one of several front-end computers, invoke Fesa Deliver:

```
Zoe> cd ./fesa/ZoeTutorial/v0
Zoe/fesa/ZoeTutorial/v0> Fesa Deliver ZoeTutorial 0
```

At this stage, the system also takes a snapshot of your C++ source code by commiting your files' changes to the central CVS repository.

2 Check that the CVS commit actually took place by browsing the equipment source-code repository which is accessible from the FESA web site.



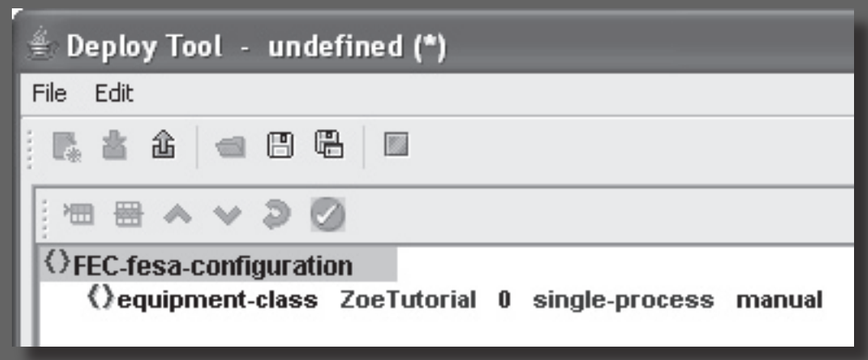
Stage Four: Deploying your class on a FEC

1. Before being able to create new instances of your newly developed equipment-class, you need to prepare one or several front-end computers that can host the class. To this end, you start the FESA Deployment Tool from the FESA website.

2. Add an entry in the desired FEC's configuration as depicted below (in addition to your class, there might already be other classes deployed on the targeted FEC, in which case you may decide to keep or remove them depending on whether they are actually needed).

3. Select the `single-process` deployment option which means both the server and real-time activities will run different threads within the same process.

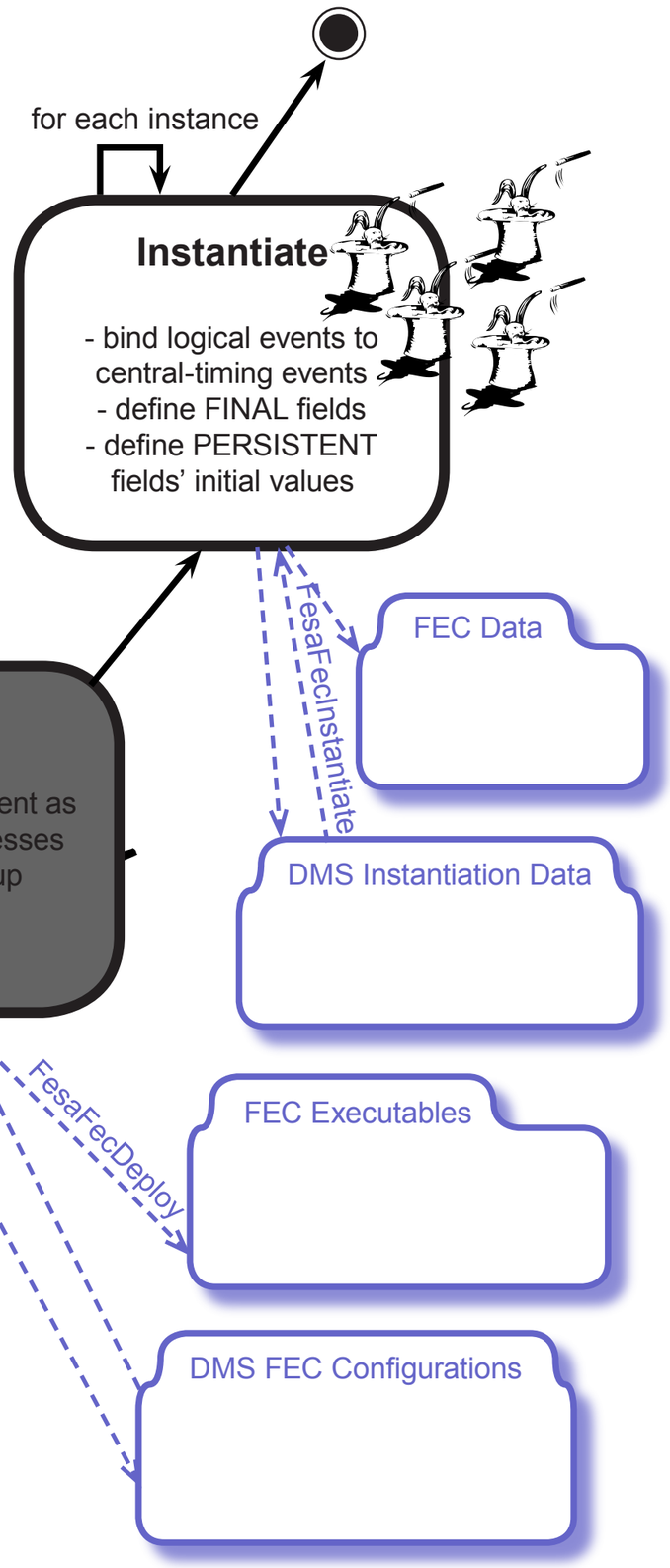
4. Select the `manual` startup mode, unless you want the class to start automatically at boot.



4. On Linux, go to the local TEST directory within your C++ development tree.

5. Run `make`:

```
Zoe> cd ./fesa/ZoeTutorial/v0
Zoe/fesa/ZoeTutorial/v0> cd TEST
Zoe/fesa/ZoeTutorial/v0/TEST> make
```

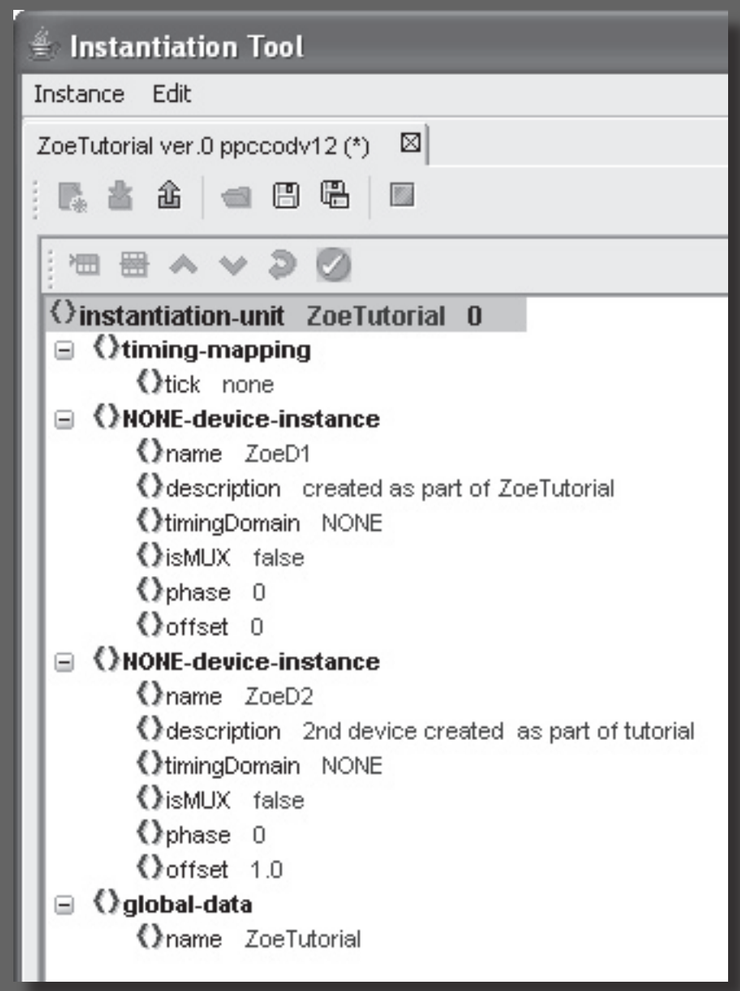


Stage Five: Instantiation

1. You are now ready to create new instances of your class. In general, each time you would need to configure a new instance each time you physically connect a corresponding new hardware device to the front-end computer (through a dedicated electronic board or through a field-bus). To this end, launch the Instantiation Tool from the FESA web site.

2. For test purposes, create two new instances named (*) `ZoeD1` and `ZoeD2` and set different offset fields for each of them, as depicted on the right.

(*) Note that for operational instances, you would have to comply with the strict CERN standard for naming devices.



3. For test purposes, you can extract instantiation data from the data-base and into your local `TEST` directory. To this end, you invoke the `Fesa Instantiate` script:

```
Zoe> cd ./fesa/ZoeTutorial/v0
Zoe/fesa/ZoeTutorial/v0> cd TEST
Zoe/fesa/ZoeTutorial/v0/TEST> Fesa Instantiate ZoeTutorial 0 ppccodv12
```

4. Everything is now ready for local testing (i.e. your local executable and the instantiation data for the front-end computer). Remote log to the targeted `ppccodv12` front-end computer and launch your equipment software:

```
Zoe> ssh ppccodv12
ppccodv12: cd /user/Zoe/fesa/ZoeTutorial/v0/TEST
ppccodv12: FesaZoeTutorialSingleProcessServer &
```

5. Congratulations! You can now remotely access devices ZoeTutorialD1 and ZoeTutorialD2 accross the middleware. To this end, you may launch the Navigator Tool from the FESA website.