Run Control

Murrough Landon – 7 November 2001

http://www.hep.ph.qmw.ac.uk/~landon/talks

Overview

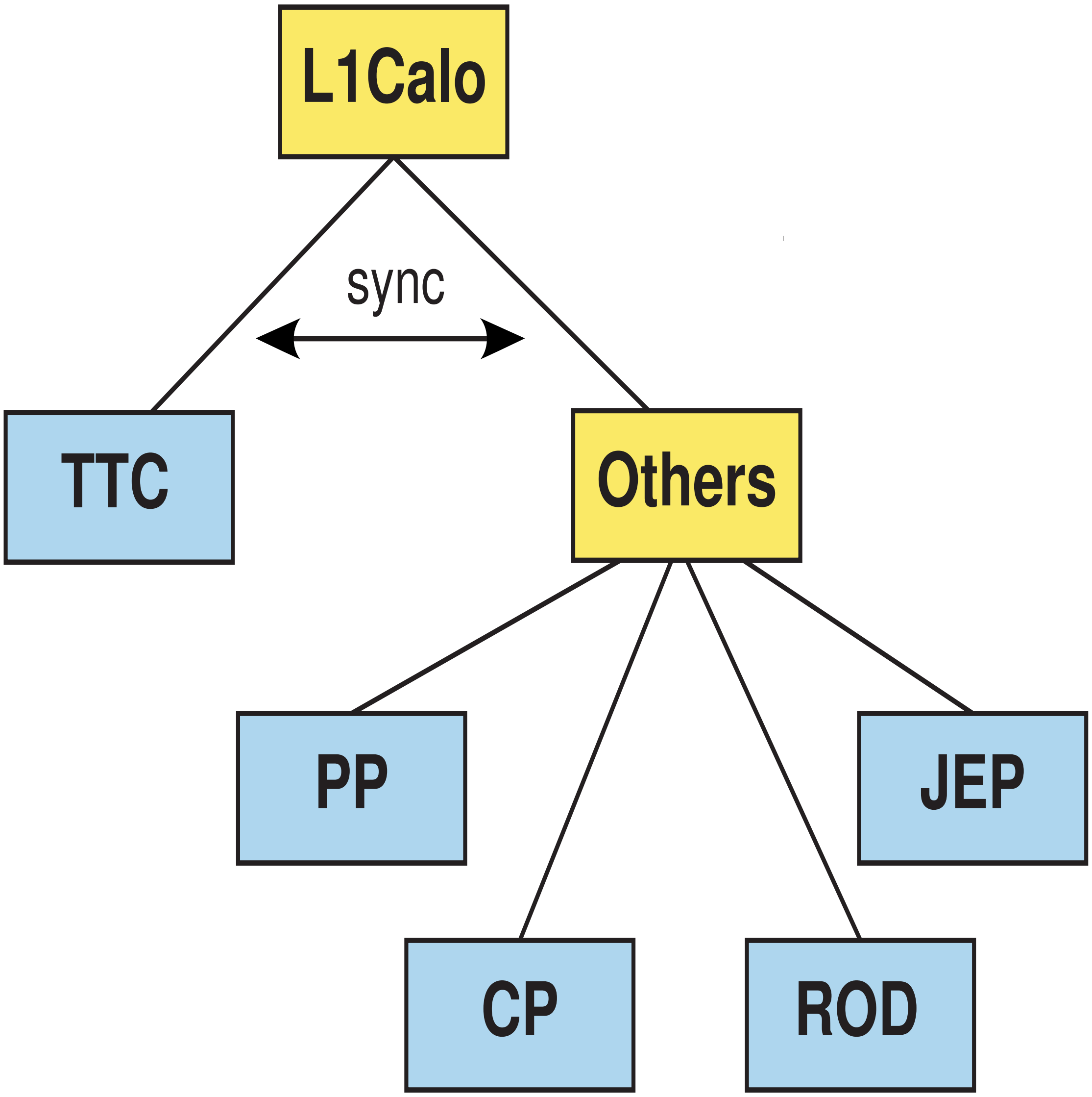
- Introduction
- Diagrams

Run Control Guide

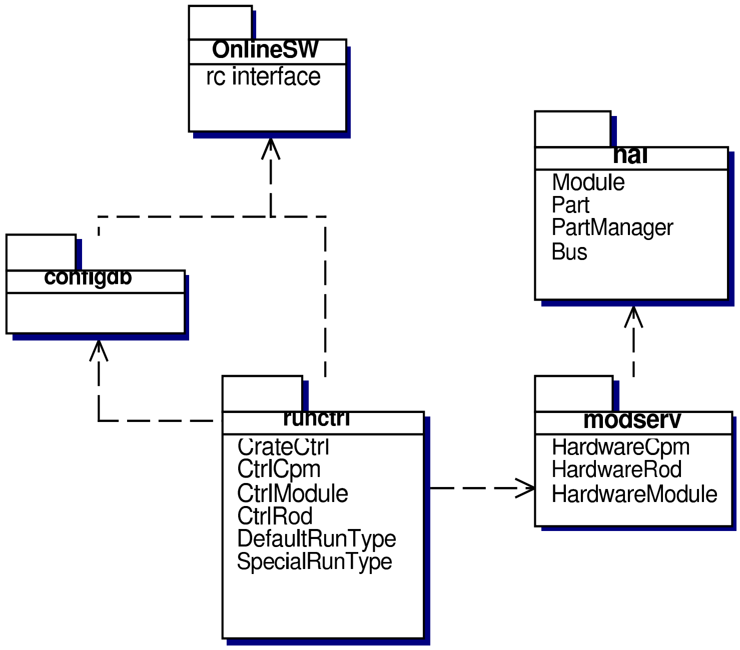
Design Principles

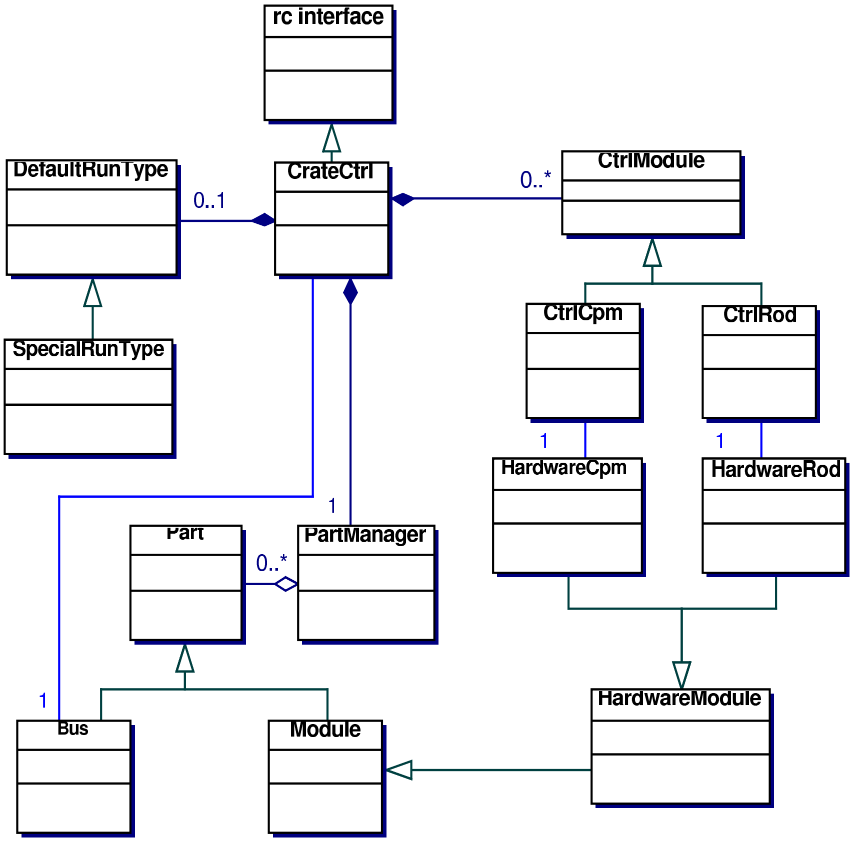
- Separate run controller per crate. Ideally the same program for all types of crate.
- Additional synchronisation controllers (very simple).
- State transition actions devolved to the modules.
- Steering of (run type dependent) transition actions in run control module classes – ie separate from hardware module classes which provide the necessary services.
- Run types implemented as subclasses of a generic base class.
- Initialise module configuration from the OKS database. But use parts files to describe the internals of the hardware modules. At least for now.

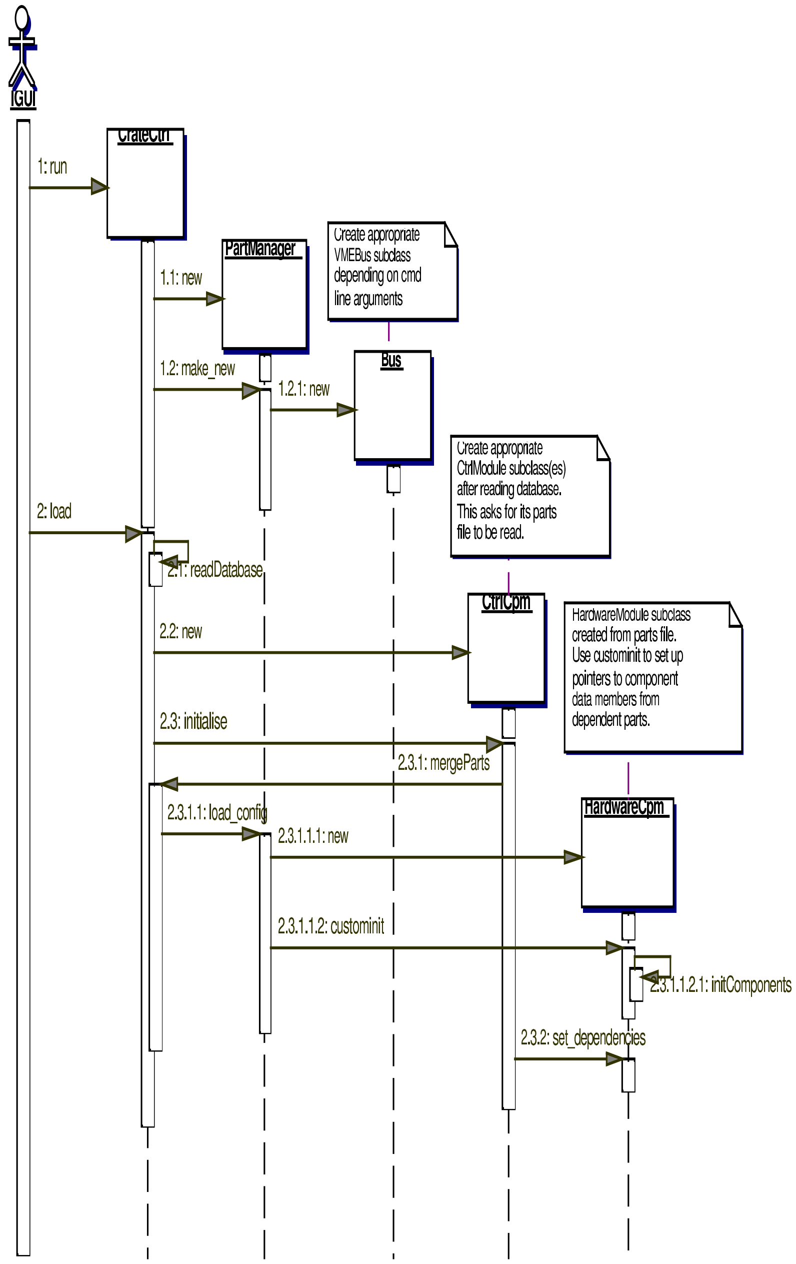
Run Controller Hierarchy for L1Calo Slice Tests



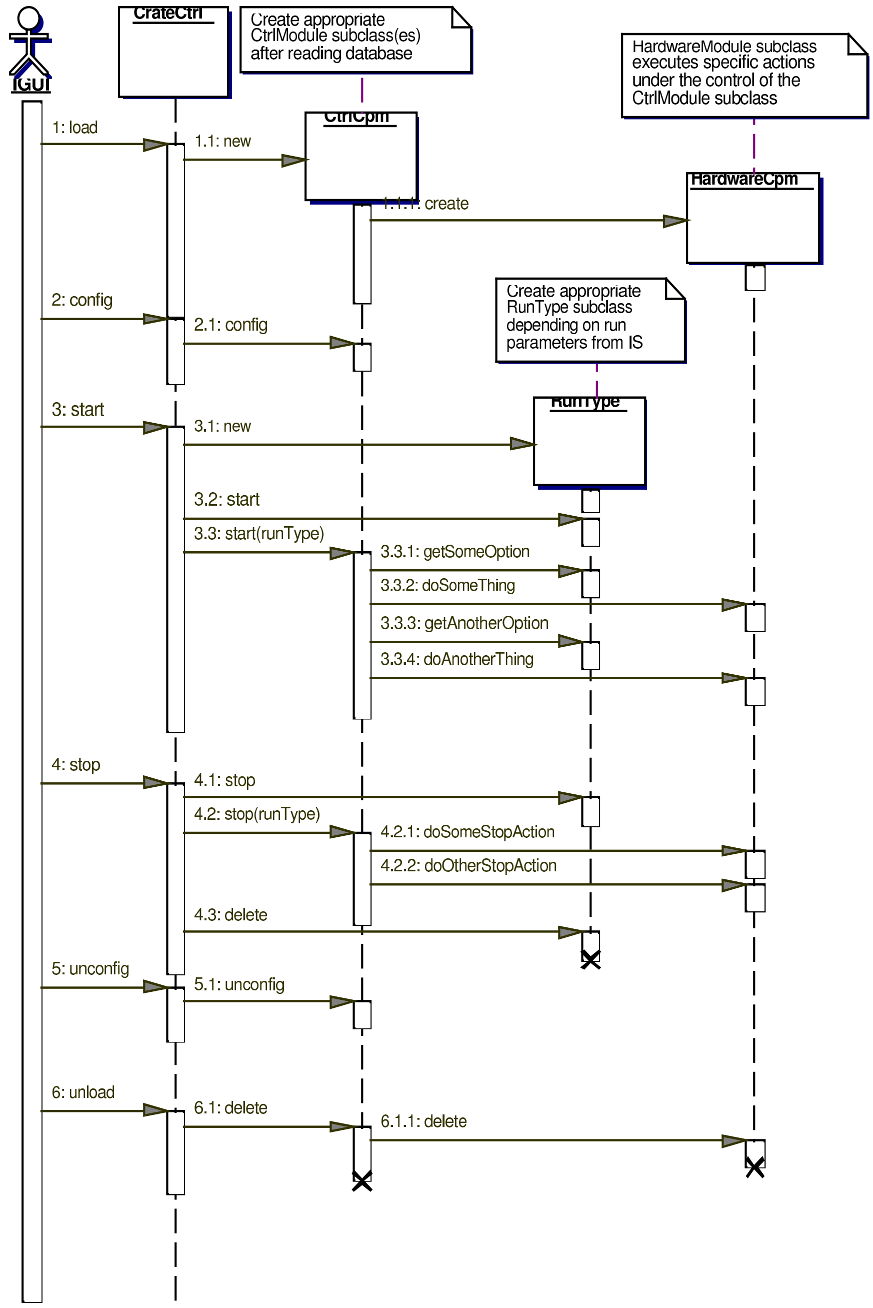
Final system probably similar, but with more than one instance of PP, CP, JEP and ROD crates

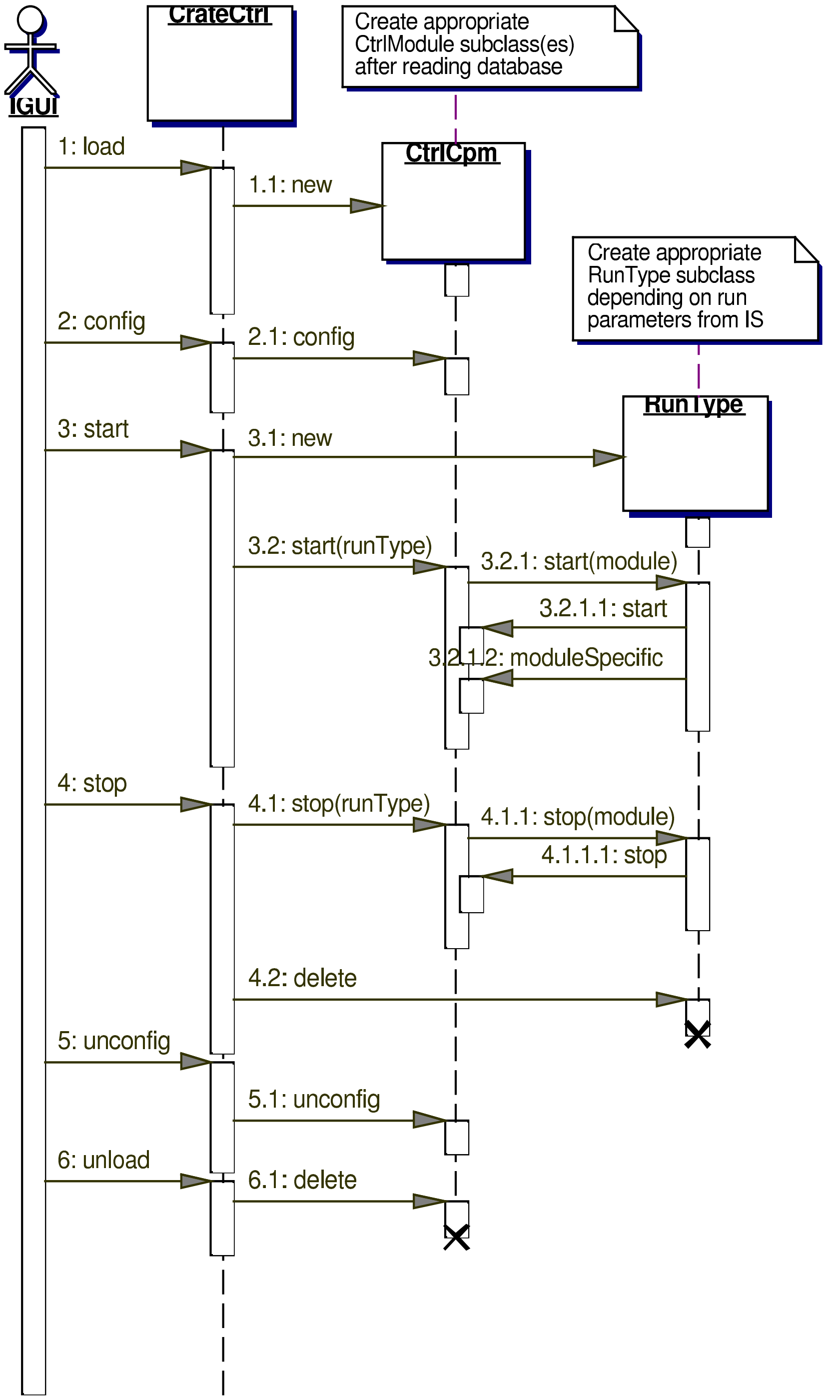

OnlineSW
rc interface
hal
Module
Part
PartManager
Bus
configdb
runctrl
CrateCtrl
CtrlCpm
CtrlModule
CtrlRod
DefaultRunType
SpecialRunType
modserv
HardwareCpm
HardwareRod
HardwareModule


rc interface
DefaultRunType
CrateCtrl
CtrlModule
0..1
0..*
SpecialRunType
CtrlCpm
CtrlRod
1
1
HardwareCpm
HardwareRod
1
Part
PartManager
0..*
1
Bus
Module
HardwareModule



GUI
CrateCtrl
1: run
PartManager
1.1: new
Create appropriate VMEBus subclass depending on cmd line arguments
Bus
1.2: make_new
1.2.1: new
Create appropriate CtrlModule subclass(es) after reading database. This asks for its parts file to be read.
2: load
2.1: readDatabase
CtrlCpm
2.2: new
HardwareModule subclass created from parts file. Use custominit to set up pointers to component data members from dependent parts.
2.3: initialise
2.3.1: mergeParts
HardwareCpm
2.3.1.1: load_config
2.3.1.1.1: new
2.3.1.1.2: custominit
2.3.1.1.2.1: initComponents
2.3.2: set_dependencies


GUI
CrateCtrl
Create appropriate CtrlModule subclass(es) after reading database
CtrlCpm
HardwareModule subclass executes specific actions under the control of the CtrlModule subclass
HardwareCpm
Create appropriate RunType subclass depending on run parameters from IS
RunType
1: load
1.1: new
1.1.1: create
2: config
2.1: config
3: start
3.1: new
3.2: start
3.3: start(runType)
3.3.1: getSomeOption
3.3.2: doSomeThing
3.3.3: getAnotherOption
3.3.4: doAnotherThing
4: stop
4.1: stop
4.2: stop(runType)
4.2.1: doSomeStopAction
4.2.2: doOtherStopAction
4.3: delete
5: unconfig
5.1: unconfig
6: unload
6.1: delete
6.1.1: delete

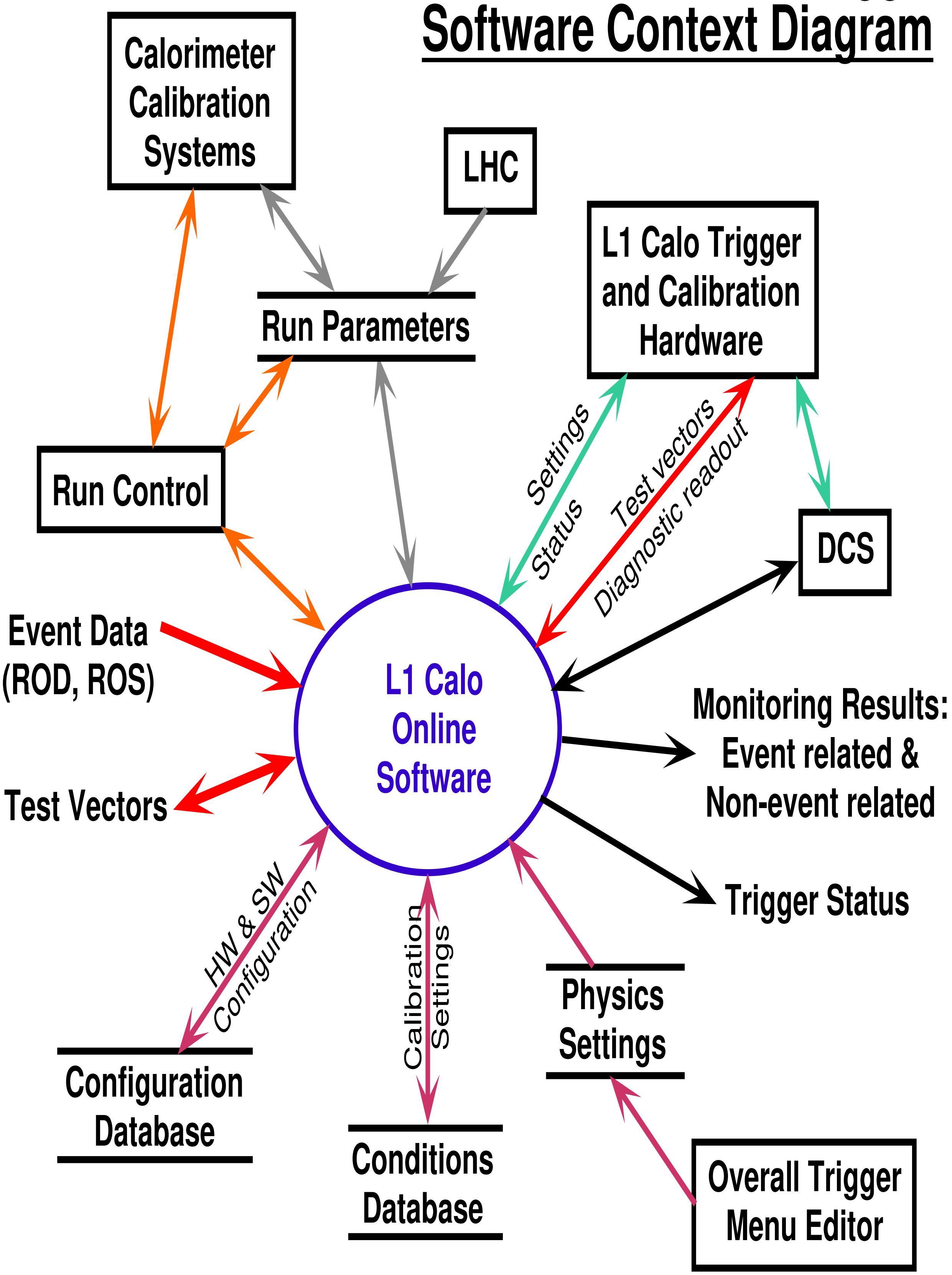

IGUI
CrateCtrl
Create appropriate
CtrlModule subclass(es)
after reading database
CtrlCpm
Create appropriate
RunType subclass
depending on run
parameters from IS
RunType
1: load
1.1: new
2: config
2.1: config
3: start
3.1: new
3.2: start(runType)
3.2.1: start(module)
3.2.1.1: start
3.2.1.2: moduleSpecific
4: stop
4.1: stop(runType)
4.1.1: stop(module)
4.1.1.1: stop
4.2: delete
5: unconfig
5.1: unconfig
6: unload
6.1: delete

Appendix

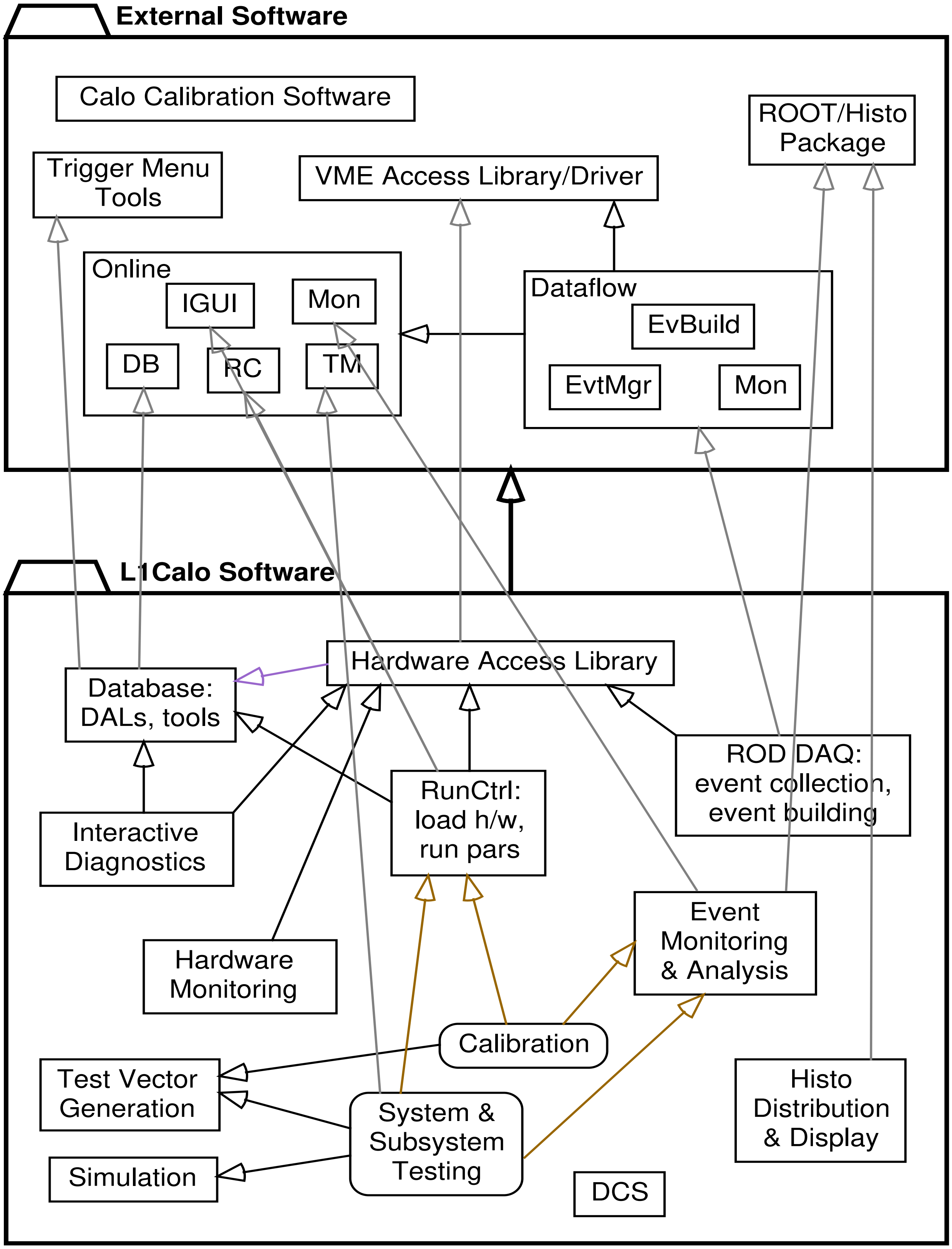
Other Diagrams

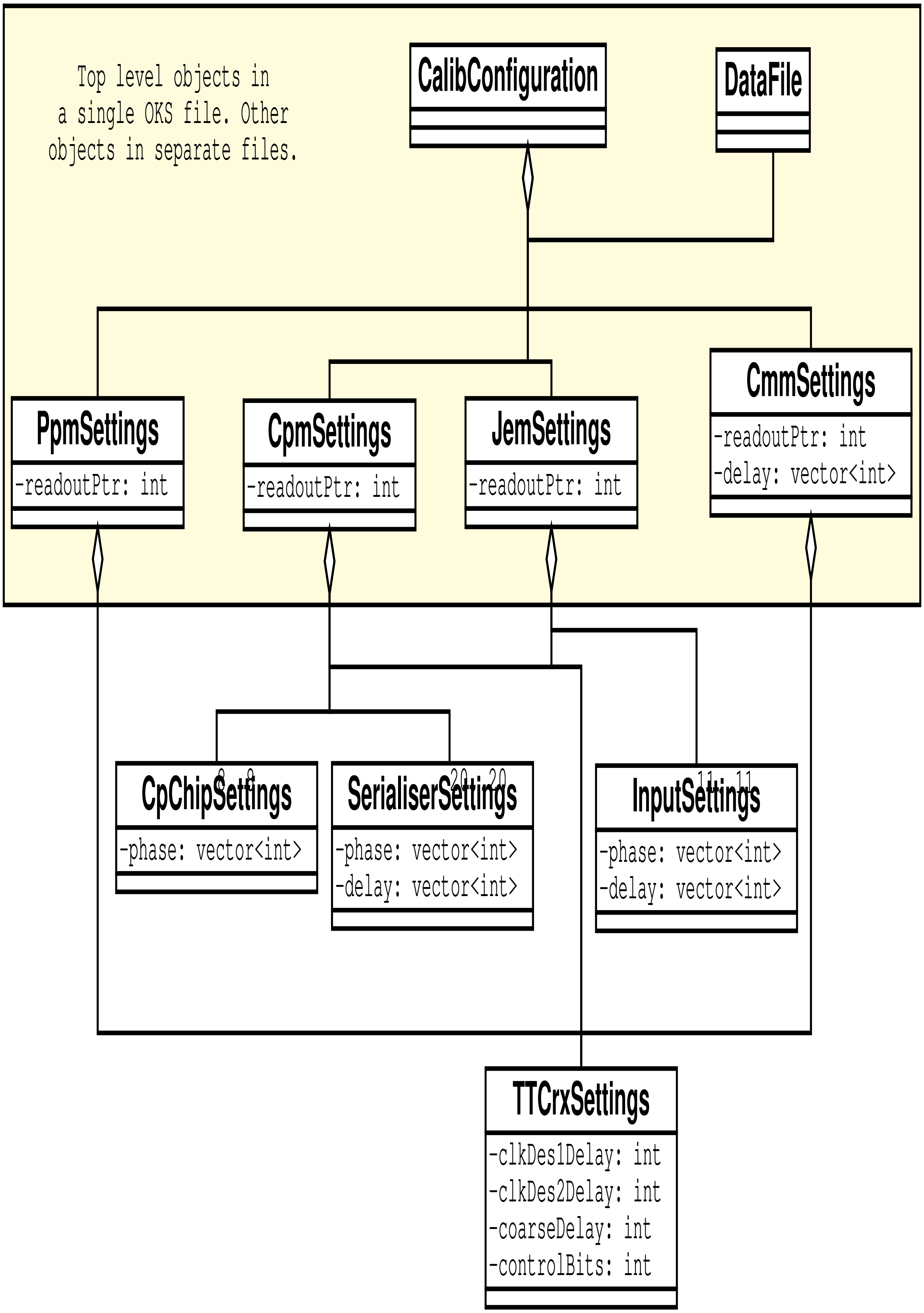
- Software context
- Software packages
- Calibration schema
- PPM calibration schema
- L1Calo trigger menu schema

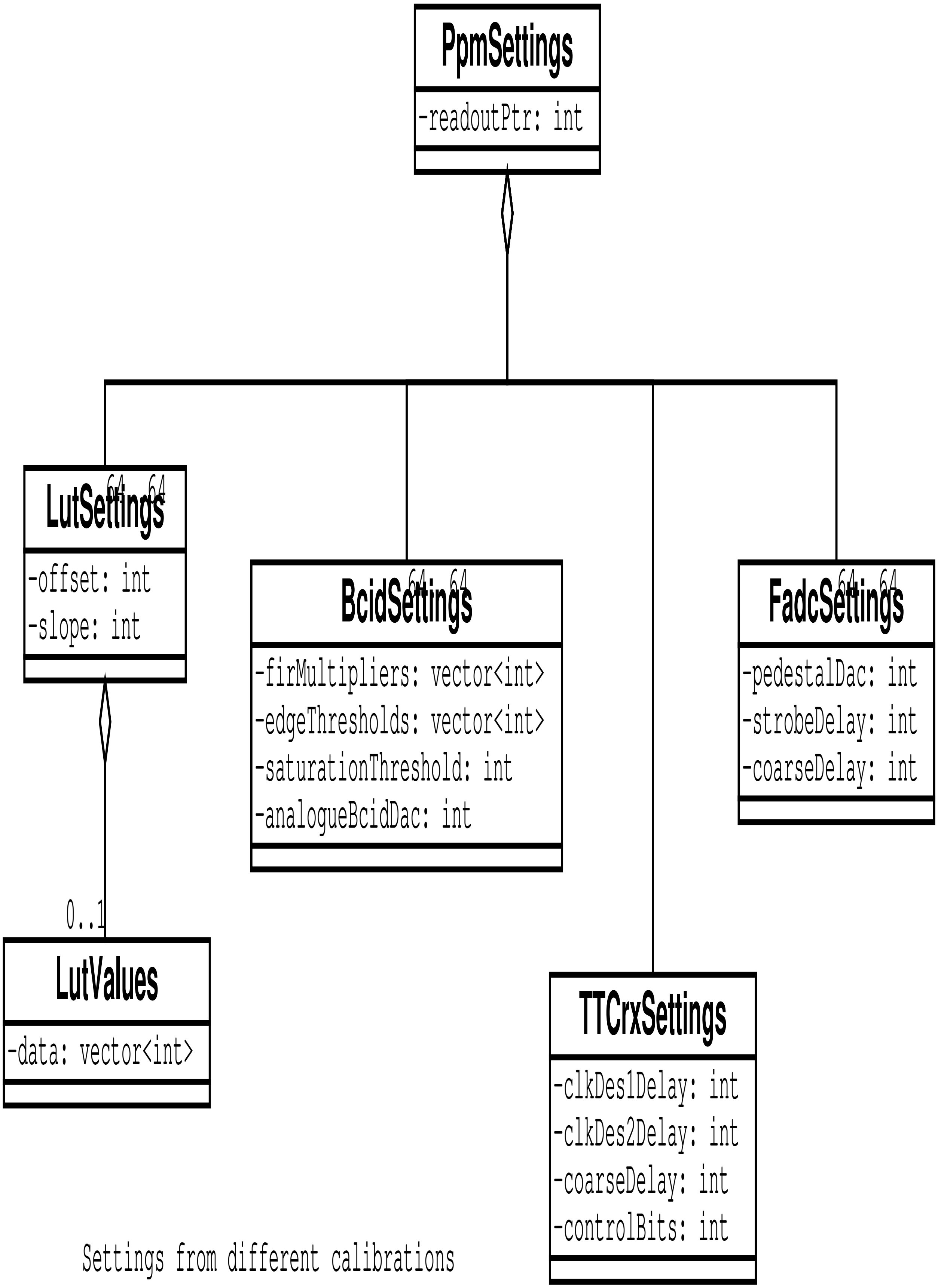
Level 1 Calorimeter Trigger Software Context Diagram



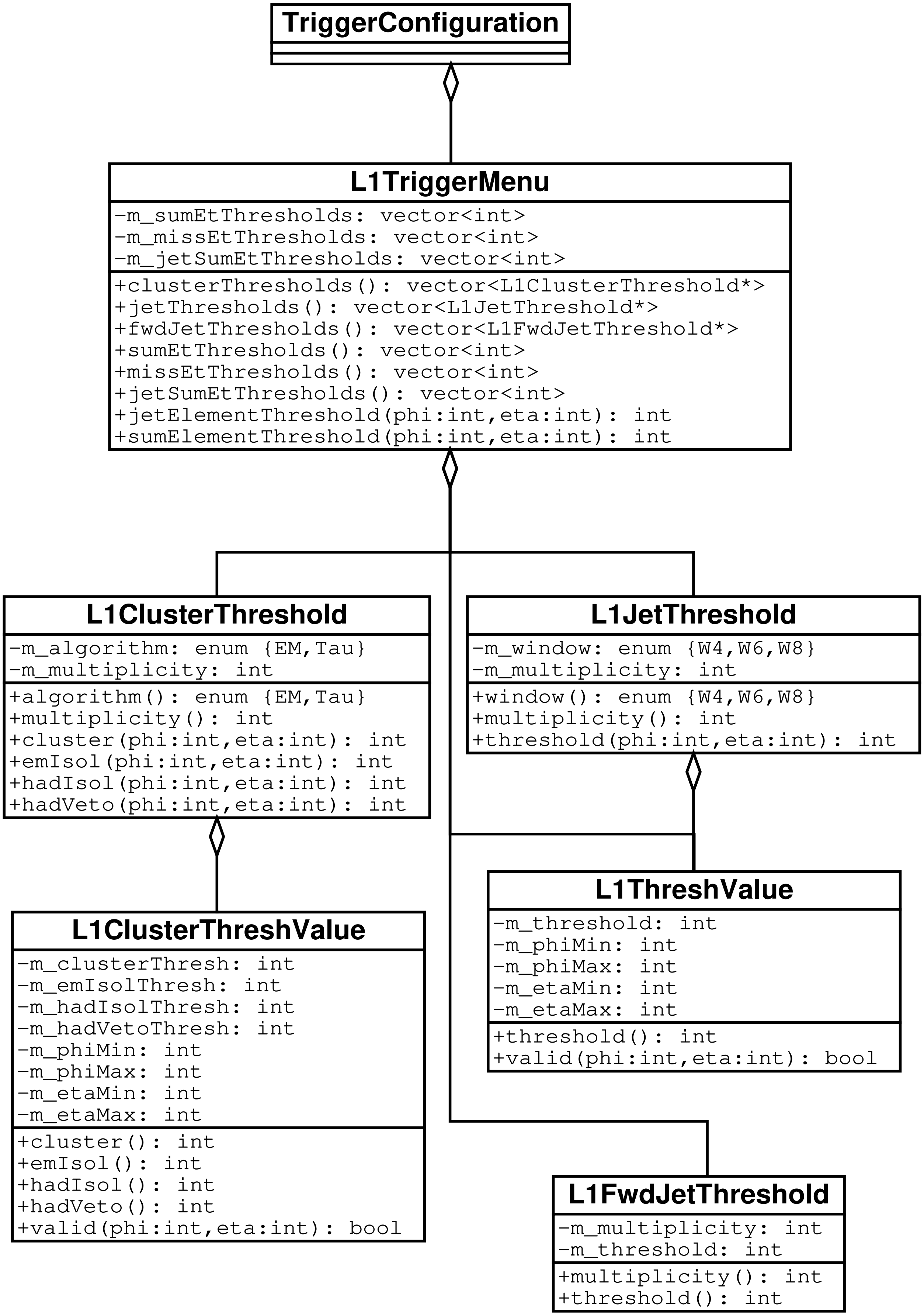
L1Calo software packages and external libraries




Top level objects in
a single OKS file. Other
objects in separate files.
CalibConfiguration
DataFile
PpmSettings
-readoutPtr: int
CpmSettings
-readoutPtr: int
JemSettings
-readoutPtr: int
CmmSettings
-readoutPtr: int
-delay: vector<int>
CpChipSettings
-phase: vector<int>
SerialiserSettings
-phase: vector<int>
-delay: vector<int>
InputSettings
-phase: vector<int>
-delay: vector<int>
TTCrxSettings
-clkDes1Delay: int
-clkDes2Delay: int
-coarseDelay: int
-controlBits: int



Settings from different calibrations
stored in separate OKS files


TriggerConfiguration
L1TriggerMenu
-m_sumEtThresholds: vector<int>
-m_missEtThresholds: vector<int>
-m_jetSumEtThresholds: vector<int>
+clusterThresholds(): vector<L1ClusterThreshold*>
+jetThresholds(): vector<L1JetThreshold*>
+fwdJetThresholds(): vector<L1FwdJetThreshold*>
+sumEtThresholds(): vector<int>
+missEtThresholds(): vector<int>
+jetSumEtThresholds(): vector<int>
+jetElementThreshold(phi:int,eta:int): int
+sumElementThreshold(phi:int,eta:int): int
L1ClusterThreshold
-m_algorithm: enum {EM,Tau}
-m_multiplicity: int
+algorithm(): enum {EM,Tau}
+multiplicity(): int
+cluster(phi:int,eta:int): int
+emIsol(phi:int,eta:int): int
+hadIsol(phi:int,eta:int): int
+hadVeto(phi:int,eta:int): int
L1JetThreshold
-m_window: enum {W4,W6,W8}
-m_multiplicity: int
+window(): enum {W4,W6,W8}
+multiplicity(): int
+threshold(phi:int,eta:int): int
L1ThreshValue
-m_threshold: int
-m_phiMin: int
-m_phiMax: int
-m_etaMin: int
-m_etaMax: int
+threshold(): int
+valid(phi:int,eta:int): bool
L1ClusterThreshValue
-m_clusterThresh: int
-m_emIsolThresh: int
-m_hadIsolThresh: int
-m_hadVetoThresh: int
-m_phiMin: int
-m_phiMax: int
-m_etaMin: int
-m_etaMax: int
+cluster(): int
+emIsol(): int
+hadIsol(): int
+hadVeto(): int
+valid(phi:int,eta:int): bool
L1FwdJetThreshold
-m_multiplicity: int
-m_threshold: int
+multiplicity(): int
+threshold(): int