Tree Correlator

Krzysztof Miernik

December 4, 2012

1 Motivation

Lets have a look at the old correlator class design.

Old correlator

```
class Correlator {
public:
    enum EConditions {INVALID_LOCATION     = 4,
                      VALID_IMPLANT        = 12,
                      VALID_DECAY          = 16,
                      BACK_TO_BACK_IMPLANT = 32,
                      DECAY_TOO_LATE       = 48,
                      IMPLANT_TOO_SOON     = 52,
                      UNKNOWN_CONDITION    = 100};
    Correlator();
    virtual ~Correlator();
    void DeclarePlots(void);
    void Init(RawEvent &rawev);
    void Correlate(EventInfo &event, unsigned int fch, unsigned int bch);
    void CorrelateAll(EventInfo &event);
    void CorrelateAllX(EventInfo &event, unsigned int bch);
    void CorrelateAllY(EventInfo &event, unsigned int fch);
    void PrintDecayList(unsigned int fch, unsigned int bch) const;
    double GetDecayTime(void) const;
    double GetDecayTime(int fch, int bch) const;
    double GetImplantTime(void) const;
    double GetImplantTime(int fch, int bch) const;

    void Flag(int fch, int bch);
    bool IsFlagged(int fch, int bch);

    EConditions GetCondition(void) const {
        return condition;
    }

private:
    (...)
};
```

The EventInfo objects passed around reveals some more.

Old correlator

```
struct EventInfo {
    enum EEventTypes {IMPLANT_EVENT, ALPHA_EVENT, BETA_EVENT, FISSION_EVENT,
                      PROTON_EVENT, DECAY_EVENT, PROJECTILE_EVENT, GAMMA_EVENT,
                      UNKNOWN_EVENT};

    EEventTypes type; ///< event type
    double time;     ///< timestamp of event
    double dtime;    ///< time since implant [pixie units]
    double energy;   ///< energy of event
    double energyBox; ///< energy depositied into the box
    double offTime;  ///< length of time beam has been off
    double foilTime; ///< time difference to foil event
    double tof;      ///< time of flight for an implant
    double position; ///< calculated strip position
    short  boxMult;  ///< numebr of box hits
```

```
    short  boxMax;   ///< location of maximum energy in box
    short  impMult;  ///< number of implant hits
    short  mcpMult;  ///< number of mcp hits
    short  generation; ///< generation number (0 = implant)
    bool   flagged;  ///< flagged of interest
    bool   hasTof;   ///< has time of flight data
    bool   hasVeto;  ///< veto detector has been hit
    bool   beamOn;   ///< beam is on target
    bool   pileUp;   ///< trace is piled-up

    unsigned long clockCount;
    unsigned char logicBits[dammIds::logic::MAX_LOGIC+1];

    EventInfo();
    EventInfo(double t, double e, LogicProcessor *lp);
};
```

Old correlator disadvantages

1. Tailored for DSSD experiments
2. Diffucult to extend
3. Will drag a lot of unecessary code if used for other purposes
4. Deeply bound into basic structures (composed into RawEvent class)

New correlator goals

Correlator: object relating events occuring in the same of different detectors at the same or at different time

1. Flexible design
2. Easy extending
3. Dynamic creation, easy modification for different experimental setups
4. Loosly bound building blocks, easy to remove if not needed
5. (More object oriented design)

2 Design

The starting point

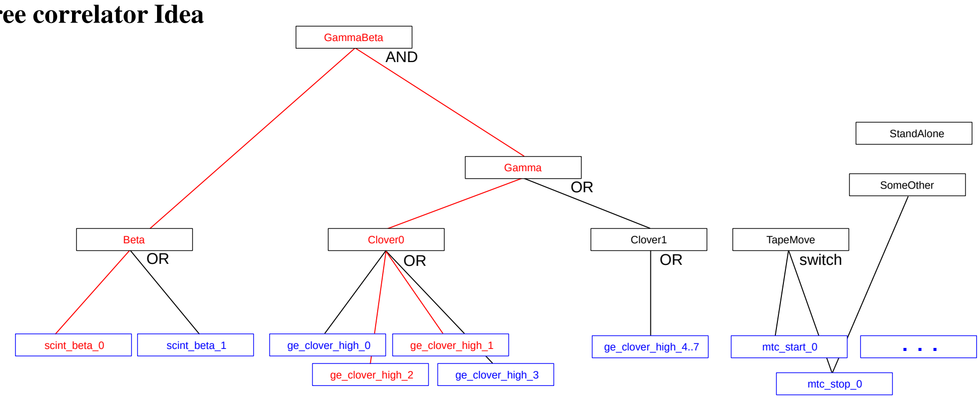
- The map2.txt file

```
MOD    CH      TYPE        SUBTYPE             TAGS
0      0-15    ge            clover_high
1      0-15    ge            clover_low
2      0-1     scint   beta
2      2       mtc     beam_start
2      3       mtc     beam_stop
2      4       mtc     start
2      5       mtc     stop
```

- Each entry in the map is related with a *basic place*.
- A *place* is an abstract object, may be related with physical objects or conditions, but may we as well an abstract condition or set of conditions.

- *Places* are characterized by Boolean state (True / False) but also may store additional information (time, energy, etc.).
- *Places* should be kept simple, each serving one simple task only (e.g. counting number of activations, performing logical operation 'or', etc.). User will be given a number of such building blocks to build a hierarchical tree scheme for his experiment.
- We will connect *places* with other *places* as we would connect analog electronics modules (e.g. noise discriminators, coincidence units, etc.).

Tree correlator Idea



3 Users

XML example

```
<?xml version="1.0" encoding="utf-8"?>
<TreeCorrelator name="root" description="LeRIBSS 2011, 2 Clovers">

    <Place type="PlaceAND" name="GammaBeta">

        <Place type="PlaceOR" name="Beta">
            <Place type="PlaceThreshold" name="beta_scint_beta_0-1"
                low_limit="10.0" high_limit="16382" replace="true"/>
        </Place>

        <Place type="PlaceOR" name="Gamma">
            <Place type="PlaceOR" name="Clover0">
                <Place type="" name="ge_clover_high_0-3"/>
            </Place>
            <Place type="PlaceOR" name="Clover1">
                <Place type="" name="ge_clover_high_4-7"/>
            </Place>
        </Place>
    </Place>

    <Place type="PlaceSwitchXAND" name="TapeMove" reset="false">
        <Place type="" name="mtc_start_0"/>
        <Place type="" name="mtc_stop_0"/>
    </Place>

    <Place type="PlaceAND" name="SomeOther">
        <Place type="" name="mtc_stop_0"/>
    </Place>

    <Place type="PlaceDetector" name="StandAlone"/>

</TreeCorrelator>
```

The naming of basic places *must* follow the map2.txt file!

Defining places - XML syntax

- Basic places are created automatically from entries in the map2.txt, their names are generated as "type_subtype_location"
- Root element should be named <TreeCorrelator>, and may have *description* attribute
- Each <Place> element has following attributes.
- Mandatory attributes:
  - *name* - required, if the last token (tokens are separated by '_') is in format X-Y,Z where X, Y and Z are integers, it will be interpreted as a list (e.g. beta_0-1,5,9-10 will create beta_0, beta_1, beta_5, beta_9 and beta_10)

Defining places - XML syntax

- Optional attributes:
  - *type* - must be one of types defined in the PlaceBuilder.cpp (see there) currently available are : PlaceDetector, PlaceThreshold, PlaceThresholdOR, PlaceCounter, PlaceOR, PlaceAND if type is not used or empty (type="") it is assumed that place already exists. In particular this is true for all basic places created from channels as defined in map2.txt
  - *replace* - if set to 'true', will replace existing place with a one defined in this element.
  - *fifo* - depth of FIFO of a place
  - *coincidence* - defines type of relation with parent (true of false)
  - *low_limit, high_limit* - required for PlaceThreshold and PlaceThresholdOR, defines threshold limits (units of calibrated energy).

Using places

- Accessing place's status

```
bool tapeMove = TreeCorrelator::get()->place("TapeMove")->status();
```

- Activating and deactivating place (if not done automatically!)
  - ```
    TreeCorrelator::get()->place("TapeMove")->activate(time);
    ```
  - ```
    CorrEventData info(time, energy);
    TreeCorrelator::get()->place("TapeMove")->activate(info);
    ```
  - ```
    TreeCorrelator::get()->place("TapeMove")->deactivate(time);
    ```
- Accessing stored information
    ```

```
TreeCorrelator::get()->place("TapeMove")->last().time;
TreeCorrelator::get()->place("TapeMove")->secondlast().time;
```

rarely needed

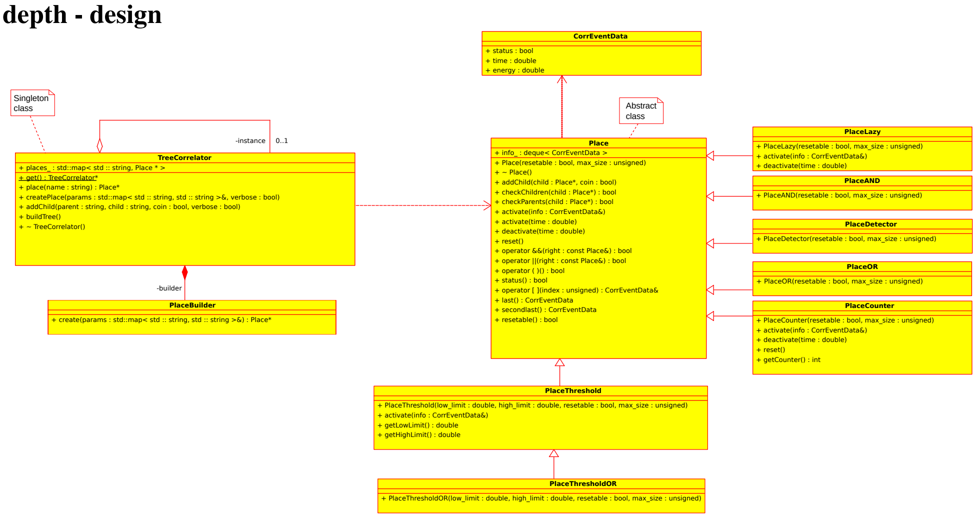
```
TreeCorrelator::get()->place("TapeMove")->info_[i];
```

The Places so far

- Place - abstract base class
- PlaceLazy - abstract, does not store multiple activation / deactivation events
- PlaceDetector - most basic (does not depend on children)
- PlaceThreshold - activates if energy between low and high thresholds
- PlaceThresholdOR - as above but depends on children status ('or' logic operation)
- PlaceCounter - counts number of activations
- PlaceOR - performs 'or' logic operation on children status
- PlaceAND - performs 'and' logic operation on children status
- PlaceSwitchANDX - switch–like behaviour

4 In depth

In depth - design



In depth - extending

- Create a new derived class in Places.hpp

```
class PlaceX : public Place {
    public:
        PlaceX(bool resetable = true, unsigned max_size = 2,
               int parX) :
            Place(resetable, max_size) {
                /*Possibly do something here*/
            }
    protected:
        virtual void check_(CorrEventData& info);
};
```

- In Places.cpp define a function *check_* (pure abstract in base class)

```
void PlaceX::check_(CorrEventData& info) {
    /* Function body */
}
```

In depth - extending

- In PlaceBuilder.hpp add a new function

```
Place* createPlaceX(std::map<std::string, std::string>& params);
```

and define it in PlaceBuilder.cpp

```
Place* PlaceBuilder::createPlaceX (map<string, string>& params) {
    bool reset = strings::to_bool(params["reset"]);
    int fifo = strings::to_int(params["fifo"]);
    int parX = strings::to_int(params["parX"]);
    Place* p = new PlaceX(reset, fifo, parX);
    return p;
}
```

- Finally in PlaceBuilder add function to the if–else loop, so XML will understand the new type.

```
if (type == "")
    return NULL;
(...)
else if (type == "PlaceX")
    return createPlaceX(params);
```

Add comment to TreeCorrelator.xml about your new parameters!