

Generalized event-driven processing with *Antelope*

February, 2007

Antelope User Group Meeting

DST, Trieste



What is Generalized Event-driven Processing?

- Consider any seismic event oriented processing that can be done after the initial location estimate
- Examples:
 - Magnitude estimation
 - Moment tensor inversion
 - Focal mechanism
 - Arrival time refinement using cross-correlation followed by relocation
 - Any other type of relocation
 - Many other examples (alarms, automatic displays, etc.)
- Many programs have been written to do specific event-driven tasks, such as **orbmag**, **dbml**, **orbampmag**, **dbampmag**, **orbgenloc**, **dbgme** and **orb_quake_alarm**.
- Much of the effort in writing these programs is in the front-end data collecting part (getting event parameters and any waveform data and metadata needed), the back-end output disposition part (writing processing results out to ORB and/or database), making this work in both real-time ORB-driven and offline database-driven environments (usually means writing two separate versions) and other features such as the ability to rapidly process multiple events in parallel for real-time systems. The actual computations do not usually depend on any of these other capabilities.

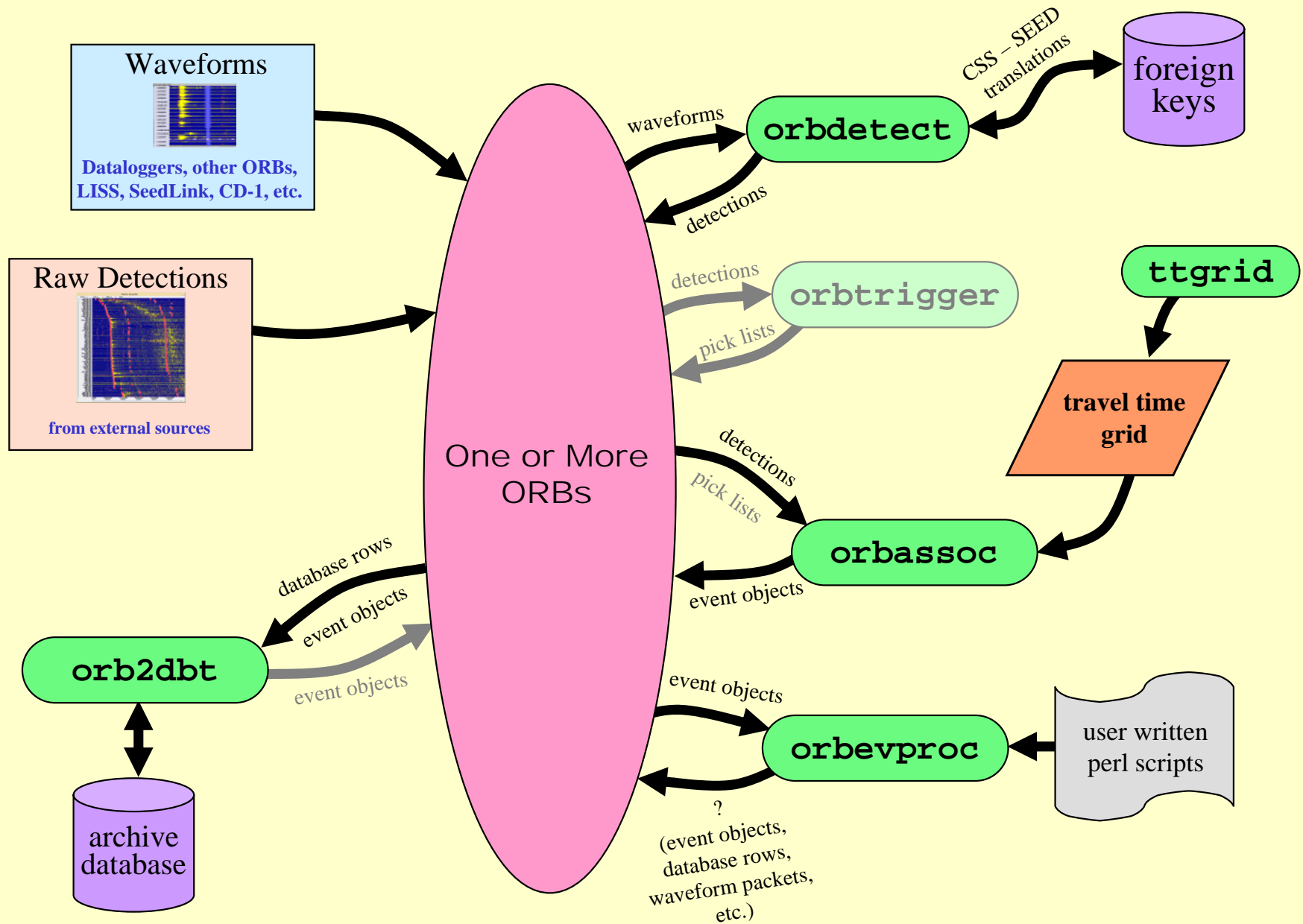
What is Generalized Event-driven Processing?

- Consider a program that acts as a “substrate” to provide all of the front-end, back-end and parallel event threading functions while allowing users to insert their own computational kernels.
- Such a program, would:
 - Work equally well in both off-line database and real-time ORB environments WITHOUT requiring ANY changes in the user-written computational kernels.
 - Provide the user computational kernels with all necessary event information and requested waveform data in a completely consistent manner regardless of where the data came from
 - Shield the user kernels from decisions and methods relating to where and how to get event information and waveform data
 - Shield the user kernels from how to dispose of output results
 - Provide the user kernels with multiple simultaneous event processing capability WITHOUT requiring threading formalisms in the user written computational code
 - Provide the user kernels with a capability for rapid computations, including early and partial results, in support of early warning applications (e.g. early magnitude estimates with incomplete data for tsunami early warning)
 - Allow for multiple different computational kernels, possibly from different authors, to be run within the same execution
 - Provide the users with the possibility of writing their computational kernels in an interpreted scripting language that would naturally open the source code and make the details of their computations available to all
 - Do all of this in an efficient and robust manner so that the end results are produced rapidly with minimum taxing of system resources

orbevproc and dbevproc

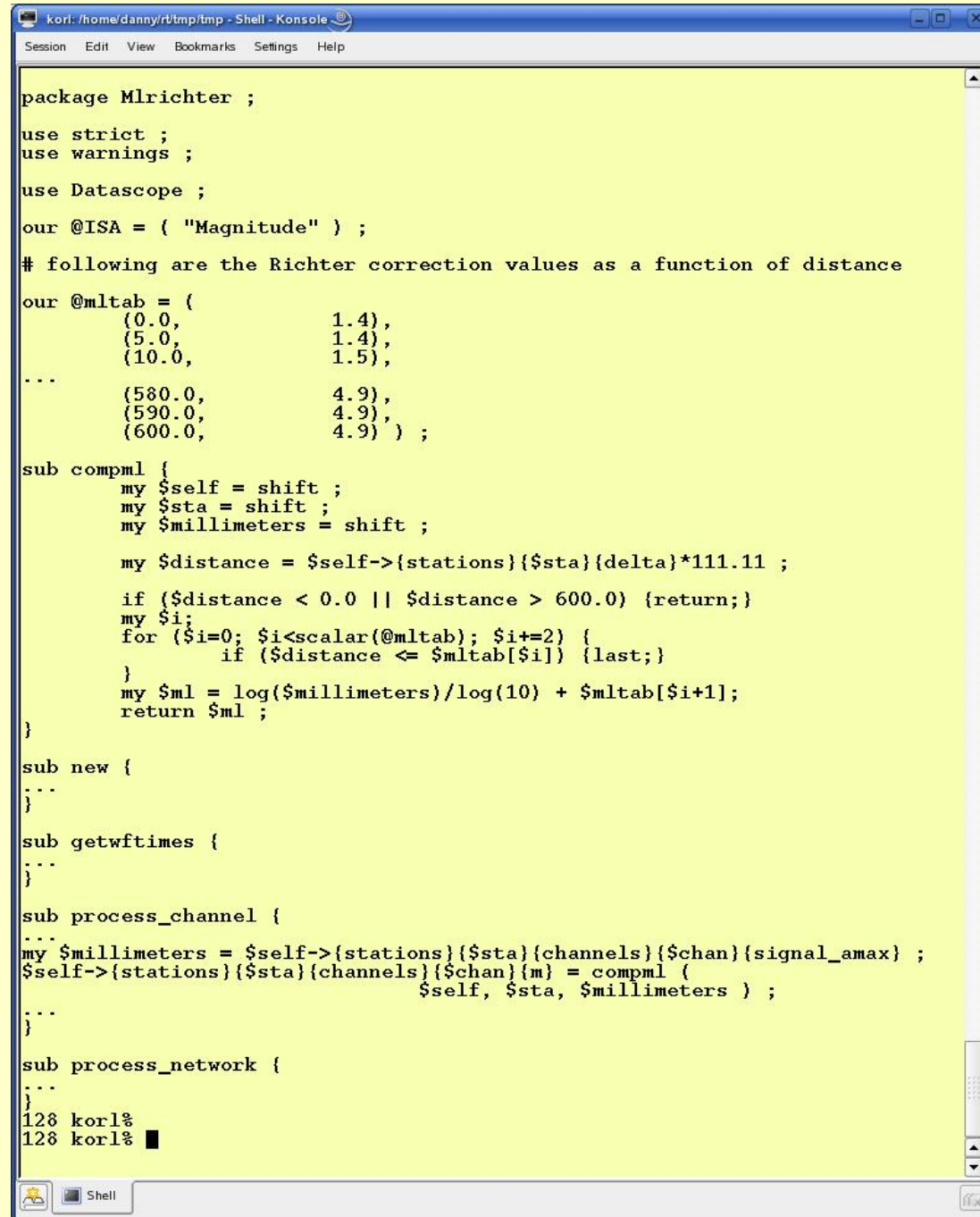
- The new programs **orbevproc** and **dbevproc** will be in the new 4.9 release of Antelope
- As with **orbassoc** and **dbgrassoc**, **orbevproc** and **dbevproc** are the same exact executable images – this insures that they operate consistently and that modifications and bug fixes to one always are applied to both
- All C code is open source and has been already installed into the Antelope contributed source code repository
- An embedded perl interpreter is used to process the user computational kernels
- In the initial release, several different magnitude computational kernels will be provided as examples
- We encourage the user community to add more computational kernels

Antelope Automated Event Processing



1. Read input event info and create temporary event database:
 - For **orbevproc**:
 1. Read event ORB object from input ORB.
 2. Create disk versions of tables in a temporary database
 - For **dbbevproc**:
 1. Read single row from **origin** table
 2. Perform joins with all other related tables (**assoc**, **arrival**, **origerr**, **emodel**, **predarr**, **netmag**, **stamag**, **wfmeas**)
 3. Unjoin into a temporary database on disk
2. For each new event create a set of event process instances by calling user-defined creation methods (**new** in perl). Pass in the temporary event database on disk plus a pointer to a meta-database (e.g. something in dbmaster). User written event processing must be implemented (currently) as perl “objects”, aka perl packages.
3. For each event, call another user written method on each of the processing object instances called **getwftimes**. The user method must return a perl hash which defines exactly which channels and time windows of waveform data are needed. Note that in **orbevproc** multiple events can be processed simultaneously. However, this is hidden from the user script level.

Perl computational package

A screenshot of a Perl script being edited in a Konsole window. The window title is 'korl: /home/danny/r/tmp/tmp - Shell - Konsole'. The script defines a package 'Mlrichter', uses 'strict' and 'warnings', and imports 'Datascopes'. It sets '@ISA' to 'Magnitude' and includes a comment about Richter correction values. A table '@mltab' is defined with distance ranges and correction values. The 'compml' subroutine calculates the magnitude correction based on distance and millimeters. The 'new' subroutine is partially shown. The 'getwftimes' subroutine is also partially shown. The 'process_channel' subroutine uses 'compml' to calculate the magnitude. The 'process_network' subroutine is partially shown. The script ends with two lines of output: '128 korl%' and '128 korl%'.

```
package Mlrichter ;  
use strict ;  
use warnings ;  
use Datascopes ;  
our @ISA = ( "Magnitude" ) ;  
# following are the Richter correction values as a function of distance  
our @mltab = (  
    (0.0,      1.4),  
    (5.0,      1.4),  
    (10.0,     1.5),  
    ...  
    (580.0,    4.9),  
    (590.0,    4.9),  
    (600.0,    4.9) ) ;  
  
sub compml {  
    my $self = shift ;  
    my $sta = shift ;  
    my $millimeters = shift ;  
  
    my $distance = $self->{stations}{$sta}{delta}*111.11 ;  
  
    if ($distance < 0.0 || $distance > 600.0) {return;}  
    my $i;  
    for ($i=0; $i<scalar(@mltab); $i+=2) {  
        if ($distance <= $mltab[$i]) {last;}  
    }  
    my $ml = log($millimeters)/log(10) + $mltab[$i+1];  
    return $ml ;  
}  
  
sub new {  
    ...  
}  
  
sub getwftimes {  
    ...  
}  
  
sub process_channel {  
    ...  
    my $millimeters = $self->{stations}{$sta}{channels}{$chan}{signal_amax} ;  
    $self->{stations}{$sta}{channels}{$chan}{m} = compml (  
        $self, $sta, $millimeters ) ;  
    ...  
}  
  
sub process_network {  
    ...  
}  
128 korl%  
128 korl% █
```

4. Read waveform data:
 - For **orbevproc**:
 1. First try to get waveform data from an archive database. Call the user written processing callbacks as each channel of waveform data has been read.
 2. After all waveform data has been read from the archive database, look for waveform data in an input ORB. Call the user written processing callbacks at user defined time intervals as waveform data is read.
 3. Quit processing and flush the processing results when either all of the requested data has been read or a user defined timeout has expired.
 4. Continue looking for new events and process new events in parallel with other event processing.
 - For **dbevproc**:
 1. Read requested waveform data from an archive input database. Call the user written processing callbacks as each channel of waveform data has been read.
 2. Event processing is strictly sequential.
5. As waveform data is read, call user written methods on each of the processing object instances; **process_channel**, whenever some more data is available for a particular station-channel, **process_station**, whenever all channels are available for a particular station, and **process_network**, whenever all channels are available for the entire network.
6. All waveform data is made available to the user methods as standard in-memory Antelope Trace objects as referenced through **trace** database table rows. Data gaps are indicated by setting the trace sample values to special gap values. There is always only one **trace** database row per channel of data. The **trace** tables row for a particular channel is passed to each user method as a Datascope database pointer. There are subroutines in the Antelope perl extensions for manipulating sample data referenced through **trace** tables.

7. User processing object instances create output results primarily by adding to the temporary event database that was passed to it when it was created. In a future modification user written object instances will also be able to specify new trace objects as output.
8. Dispose of output results:
 - For **orbevproc**:
 1. Convert temporary event database, along with new tables and modifications of existing tables, back into an event ORB object and write it to an output ORB.
 2. Lots of changes to **orb2dbt** to deal with properly merging changes to existing origins back into the archive database.
 - For **dbevproc**:
 1. Directly merge temporary event database, along with new tables and modification of existing tables, back into as output archive database using the same procedures as **orb2dbt**.

What is an event ORB “object”?

A complete properly indexed temporary database, containing only one origin and all relevant linked other tables, encapsulated into a single parameter file ORB packet usually with srcname **/pf/orb2dbt**

```
korl: /home/danny/r/gsn/sorb
58 korl%
58 korl%
58 korl%
58 korl%
58 korl% orb2pf -number 1 -start OLDEST -select /pf/orb2dbt ruper:scdemo -

/pf/orb2dbt 1263 2/15/2007 17:32:31.413
arrivals &Literal{
MONP 1171560724.70000 1 2007046 -1 -1 BHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
TRO 1171560730.02500 2 2007046 -1 -1 SHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
PFO 1171560731.02500 3 2007046 -1 -1 BHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
LVA2 1171560729.47500 4 2007046 -1 -1 BHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
SND 1171560732.00000 5 2007046 -1 -1 BHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
FRD 1171560731.30000 6 2007046 -1 -1 BHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
KNW 1171560734.22500 7 2007046 -1 -1 BHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
CRY 1171560733.07500 8 2007046 -1 -1 BHZ P - 0.100 -1.00 -1.00 -1.00 -1.00 -1.00 -1.00 -1.000 2
}
assocs &Literal{
1 2 MONP P 9.99 2.271 121.62 302.84 0.032 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 5
2 2 TRO P 9.99 2.658 133.24 314.46 0.029 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 6
3 2 PFO P 9.99 2.740 134.16 315.41 -0.089 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 9
4 2 LVA2 P 9.99 2.631 128.75 310.04 -0.148 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 1
5 2 SND P 9.99 2.793 131.37 312.70 0.153 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 3
6 2 FRD P 9.99 2.749 130.61 311.93 0.061 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 1
7 2 KNW P 9.99 2.963 132.62 314.00 0.043 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 4
8 2 CRY P 9.99 2.880 130.14 311.54 0.027 d -999.0 - -999.00 - -999.0 1.000 taup/iasp91 -1 6
}
emodel &Literal{
2 3.8561 4.38894 4.50075 4.35157 1171560750.67205
}
origerr &Literal{
2 14214.3916 10134.7653 239474543.8515 3417812.8965 -11951.4452 899262.9430 -854743.4618 105712.57832
}
origin &Literal{
31.6814 -114.1501 1.6044 1171560686.10691 2 -1 2007046 8 8 -1 49 4 - - -999.0000 f -999.00 8
}
predarrs &Literal{
1 2 1171560724.66790 13.75 121.62 -1.00 302.84 -44.2 1171560750.67481
2 2 1171560729.99630 13.75 133.24 -1.00 314.46 -44.2 1171560750.67567
3 2 1171560731.11445 13.75 134.16 -1.00 315.41 -44.2 1171560750.67602
4 2 1171560729.62257 13.75 128.75 -1.00 310.04 -44.2 1171560750.67618
5 2 1171560731.84727 13.75 131.37 -1.00 312.70 -44.2 1171560750.67636
6 2 1171560731.23872 13.75 130.61 -1.00 311.93 -44.2 1171560750.67652
7 2 1171560734.18215 13.75 132.62 -1.00 314.00 -44.2 1171560750.67667
8 2 1171560733.04835 13.75 130.14 -1.00 311.54 -44.2 1171560750.67683
}
59 korl% █
```

What is an event ORB “object”?

- At a minimum there must be a single row of an **origin** table encapsulated into a pf “Literal” string. Event objects with only an origin row are treated by **orb2dbt** as external catalog events to be associated with the existing events in the archive database. These type of event objects are produced by **dborigin2orb**.
- In order to promote detections to arrivals for a fully defined origin, such as what we would want to do with **orbassoc** output, the **assoc** and **arrival** tables must also be defined in the event object.
- Additional tables relating to errors and modeling, such as **origerr**, **emodel** and **predarr**, may also be defined, but these are optional.

notes

- All input and output of general parameters to and from the user written perl objects are done through embedded perl hashes.
- The user written perl objects generally do not know if they are being run from **orbevproc** or **dbevproc**.
- The details of how to deal with partial results have yet to be worked out