Log Analysis Tools

http://MODMED.minmaxmedical.com

(ANR-15-CE25-0010)

Agenda

Log analysis goals *(5')*	Fabrice Bertrand, Blue Ortho
Verifying complex properties *(20')*	Yoann Blein, L.I.G.
Merging, classifying, relating events *(10')*	Manon Linder, MinMaxMedical
Improving trace points *(10')*	Vivien Delmon, MinMaxMedical
Exploring large logs interactively *(5')*	Arnaud Clère, MinMaxMedical
Developers survey	

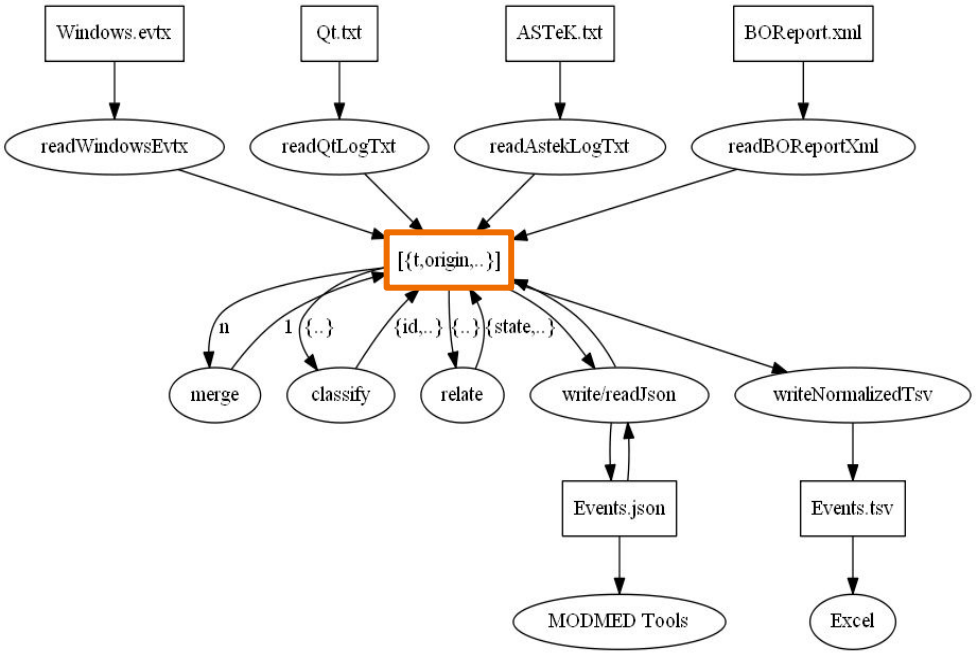
Log analysis goals

See
bertrand_Log_Analysis_Goals_(fr).pdf

Verifying complex properties

See
blein_An_Overview_of_the_Monitoring_Engine.pdf

Merging, Classifying, Relating Events



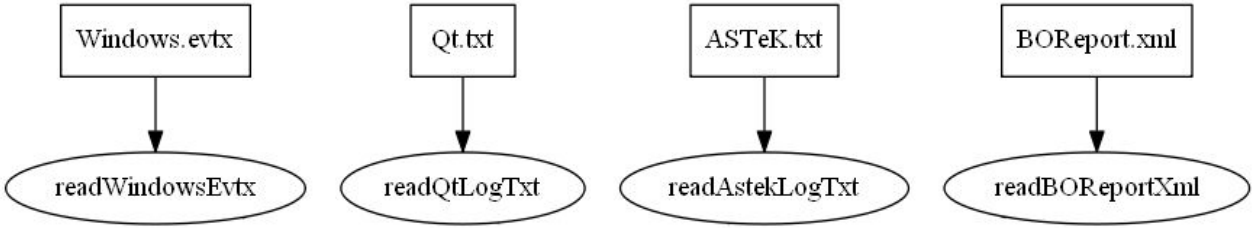
Log files:

- Several sources
- Different structures

But, possess different informations

- One file
- One **standard format**

Merging:



Log formats:

- Xml
- Structured string
- Unstructured string

Standard format with JSON style:

- Each event:
  - {time, source, severity, data}
- List of events:
  - [{time, source, severity, data}, {}, ..., {}]

Merging:

- Enforce ‘time’ monotonicity between sources
- Maintain each source’s sequence order

Classifying: 1/2

All existing **events** are not classified and the “**data**” attribute is **unstructured**.

Classification = recognize pattern in each event

- **Matching** unstructured string with known patterns
- **Extracting** useful data and create new **user-defined attributes**

Assign each event matching a specific pattern **a unique “Id”**

- Contain specific attributes
- Data can be used confidently for analysis

Classifying: Field names and values (tags) 2/2

MODMED project will propose standard field names and values (tags)

Conservative rules:

- Do not rely on case to distinguish names (but_use_it_IfYouLike)
- Begin with a letter [a-zA-Z] (‘_’ is legal but reserved)
- Continue with as many letters [a-zA-Z], digits [0-9], or ’_’

(compatible with C++, Python, Javascript, Xml, CEE and much more)

Relating Events:

Relate events = recognize patterns in the sequence of events

- Simplify the sequence
- Complement each event with data from related events
  - For example, the entry/exit of a state

Improving Trace Points

1. Use Qt categories consistently with architecture (reuse namespaces)
2. Use semi-structured event data consistently
    1. Adopt and extend a dictionary of field names and values (tags)
    2. Define Bind<_,ImportantDataType>
3. Trace input data
    1. At user and hardware interfaces
    2. At module interfaces
4. Add tracepoints where you may have raised an exception
5. [Trace functions enter/exit with input/output arguments]
   (not easy for now)
6. [Group tight trace points inside a ScopedMessage]
   (not in open source version)

Exploring Large Logs Interactively 1/3

Python code for normalizing semi-structured logs:

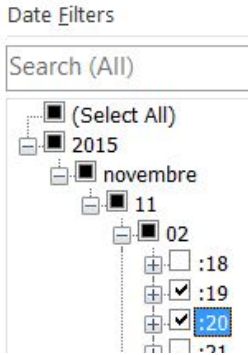
Unordered event dictionary → Ordered fields tuple

1. List important field names in the order they should appear
2. Keep all other data in *'other'* field
3. Push less important metadata to the right

writeTSV('time', 'appState', 'severity', 'id', *'other'*, 'source', 'pid', 'tid'**)**

Exploring Large Logs Interactively 2

MS Excel as a large log viewer
(up to 1M events, 250MB+)

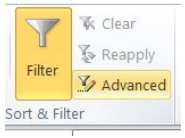


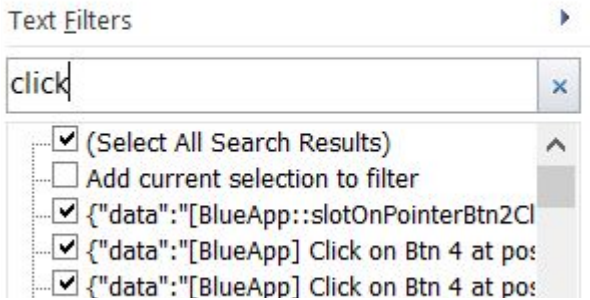
- ❏ Visualising **time** → → local ↕ ↓ relative
- ❏ Visualising **appState** → →
- ❏ Visualising **severity** ⤵

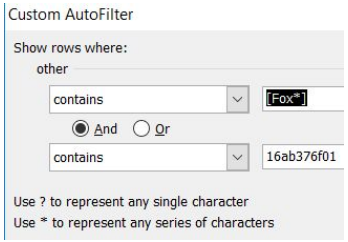
time	timeDelta	appState
2015-11-11 02:20	02:53,053	.AcquiHipCenter.Wait
2015-11-11 02:34	16:52,945	.AcquiHipCenter.Wait
2015-11-11 02:34	16:53,132	.AcquiHipCenter.Wait
2015-11-11 03:06	48:00.945	.AcquiHipCenter.Wait

Exploring Large Logs Interactively 3/3

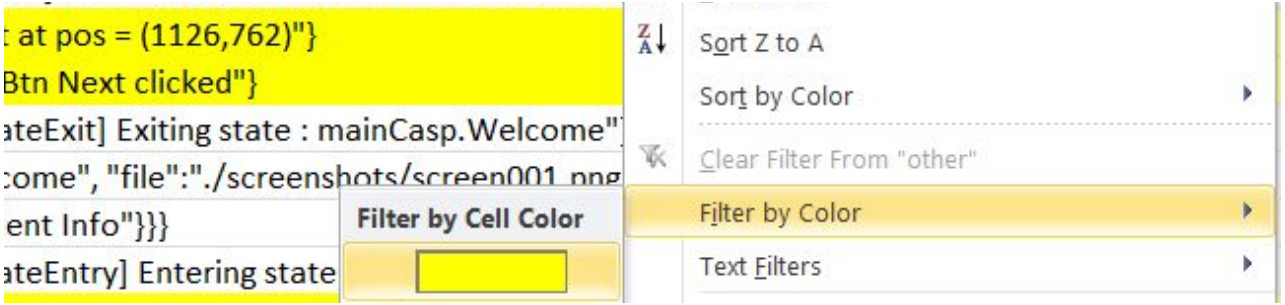
- **Filtering** (severity, category/id, data)







- **Bookmarking** with colors



Developers Survey

Please answer our survey and see the results:

http://goo.gl/fAZqhY