LOCAL MONITORING AND CONTROL SYSTEM FOR THE SKA TELESCOPE MANAGER:

A KNOWLEDGE - BASED SYSTEM APPROACH FOR ISSUES IDENTIFICATION WITHIN A LOGGING SERVICE

M. Di Carlo[1], M. Dolci[1], G. M. Le Roux[2], R. Smareglia[3], P. S. Swart[2]

[1] INAF - Osservatorio Astronomico di Teramo, [2] SKA South Africa, Cape Town, [3] INAF - Osservatorio Astronomico di Trieste

Abstract. The SKA Telescope Manager (SKA.TM) is a distributed software application aimed to control the operation of thousands of radio telescopes, antennas and auxiliary systems (e.g. infrastructures, signal processors, ...) which will compose the Square Kilometre Array, the world's largest radio astronomy facility currently under development. SKA.TM, as an "element" of the SKA, is composed in turn by a set of sub-elements whose tight coordination is ensured by a specific sub-element called "Local Monitoring and Control" (TM.LMC). TM.LMC is mainly focussed on the life cycle management of TM, the acquisition of every network-related information useful to understand how TM is performing and the logging library for both online and offline sub-elements. Given the high complexity of the system, identifying the origin of an issue, as soon as a problem occurs, appears to be a hard task. To allow a prompt diagnostics analysis by engineers, operators and software developers, a Knowledge-Based System (KBS) approach is proposed and described for the logging service.

What is a Log?

- Simplest possible storage abstraction
  - *It says what happened and when*
- For a distributed application, it can be the only way to find out an error
- Usually log files are written in a natural language: this does not allow to reason programmatically (using source code) about those informations

Information to log

- From an high level point of view the kind of applications like **SKA TM** define a logical network of interactions which it is composed by different nodes that interact each other and some of them act as coordinator or controller (at least for the online part of the system)
- It is important to log:
  - *Node identification,*
  - *Node signal declaration,*
  - *Node interactions,*
  - *Actions and loops.*

In a network, a node is a connection point, either a redistribution point or an end point for data transmissions. In general, a node has programmed or engineered capability to recognize and process or forward transmissions to other nodes.

An information usable by a node in order to control the behaviour of another one or its behaviour in function with the one from another node

In a distributed environment

- the entities which make up an application are active (processes or agents)
- the interactions are based on message exchange mechanism
- the process life time is connected to the application life time; the life time of an agent are usually independent from life time of a specific application
- the logical architecture can be set by different patterns: client-server, peer to peer, etc.
- the middleware realize the physical and logical connection between entities (subsystem, service, object, component, process, agent, etc.)

Knowledge - based system

- is a computer program that reasons and uses a knowledge base to solve complex problems
- It is composed by two types of sub-systems:
  - a knowledge base (*facts about the world*)
  - an inference engine (*logical assertions and conditions about the world*)
- To build it, it is possible to use a formal language like ***Prolog***

Prolog formalism for a logging language

- A *fact* must start with a *predicate* (which is an *atom*) and end with a *fullstop*. The predicate may be followed by *one or more arguments* (separated by commas) which are enclosed by *parentheses*.
- The arguments can be atoms (in this case, these atoms are treated as constants), numbers, variables or lists.
- *predicate(arg1, arg2, ..., argN).*

- Prolog can be seen as a language for database queries
- In a relational database a *tuple* is a generic element of a relation with attributes

employee(Id,LastName,FirstName,DeptNumber,Salary)

```
employee(1,'Carter','John',1, 25500).
employee(2,'Deere','Betty',3,28500).
employee(3,'Smith','Matt',3,27000).
...
```

department(DeptNumber,Name,MgrID)

```
department(1,'Functional',1).
department(3,'Customer',2).
department(5,'Process',18).
...
```

```
well_paid_employee(First,Last) :- employee(_Num,Last,First, _Dept,Sal), Sal > 28000.

customer_manager(First,Last) :- department(_Deptno,'Customer',MgrID),
employee(MgrID,Last,First,_Dept,_Sal).
```

Interface

Enum | String | Array of string

```
logger.log(level, keyword-predicate, arguments);
```

- Automatically translate log informations into prolog facts solving the disadvantage, for developers, to study and use prolog.
- The only thing they have to know is to choose a list of words representing the informations needed to store
- There's still a need to learn the prolog language in order to gain full advantage from the use of the declarative approach.

Tango Counter example

Commands

- **PlusOne**: increase the counter by one;
- **MinusOne**: decrease the counter by one;
- **Read**: read the value of the counter;
- **SetMaxValue**: set the max value for the counter (the counter will start from 0 and will never reach the maximum value but only the max – 1);
- **SetCounterConsumer**: set the name of the device which will use it (for logging purpose);
- **Reset**: set the counter value to 0.

Log expected

Node identification entity(entity-name, entity-type).

*****

entity(counter, 'LRU').

Node signal declaration declares(entity-name, signal).

*****

declares(counter, '+').
declares(counter, '-').
declares(counter, reset).
declares(counter, read).

Node interactions relation-word(emitter-name, receiver-name, signal, ...).

*****

interaction(consumer, counter, '+').

interaction(S, X, Y, Z), S@<date(2015,5,15,10,50,0,31,-,-).

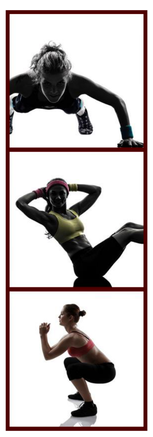
```
entity(date(2015,5,15,10,50,0,0,-,-),counter, 'LRU').
declares(date(2015,5,15,10,50,0,0,-,-),counter, '+').
declares(date(2015,5,15,10,50,0,0,-,-),(counter, '-').

entity(date(2015,5,15,10,50,0,0,-,-),clock, 'LRU').

interaction(date(2015,5,15,10,50,0,10,-,-), clock, counter, '+').
...
interaction(date(2015,5,15,10,50,0,90,-,-), clock, counter, '+').
interaction(date(2015,5,15,10,50,0,100,-,-), clock, counter, '-').
interaction(date(2015,5,15,10,50,0,110,-,-), clock, counter, '-').
interaction(date(2015,5,15,10,50,0,130,-,-), clock, counter, '-').
interaction(date(2015,5,15,10,50,0,140,-,-), clock, counter, '+').
interaction(date(2015,5,15,10,50,0,160,-,-), clock, counter, '+').
interaction(date(2015,5,15,10,50,0,170,-,-), clock, counter, '+').
...
event(date(2015,5,15,10,51,0,0,-,-), counter, minute).
```

Simple query as Sql query

Complex query: count(signal «+») – count(signal «-»)

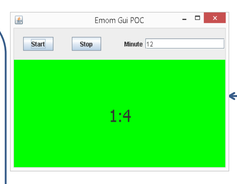
findall([], interaction(D,X,Y,'+'), L), length(L, N), findall([], interaction(D, X,Y,'-'), L1), length(L1, N1), Tot is N-N1.

Emom example



- Emom, in the context of a gym, means "every minute on the minute" and it is a technique for training for which a gymnast has to make an exercise every minute in less than a minute.
- Usually in the even minute there's an exercise and in the odd minute another one
- There are many app (usually for smartphone) which indicate with a colour weather the minute is odd or even

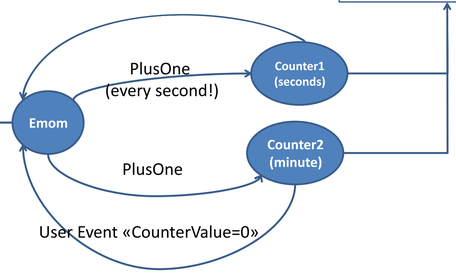
User Event «Odd»



User Event «Even»

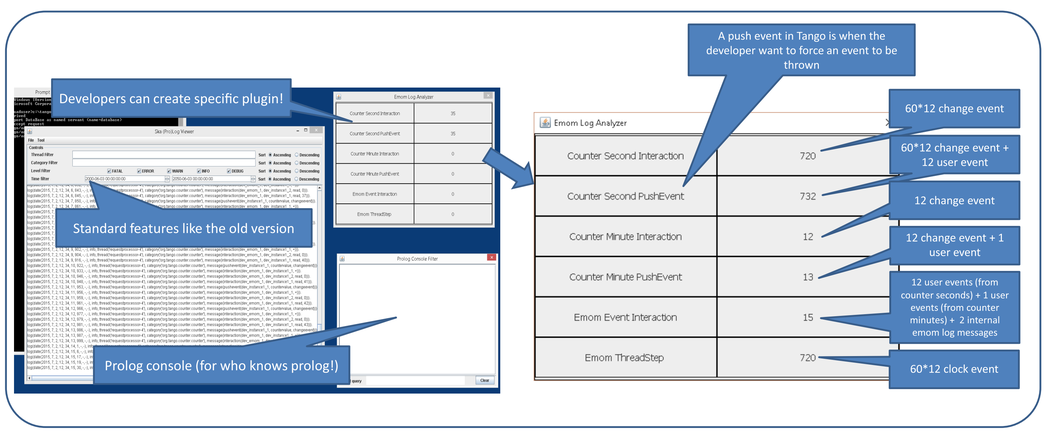
User Event «CounterValue=0»

Change Event CounterValue



1. The emom device sends a «PlusOne» command every second to the first counter (for second)
2. A user event is generated where the counter second reset its value
3. When happens a «PlusOne» command to the second counter (for minute)
4. A user event is generated where the counter minute reset its value
5. If minute is «odd», the emom throw the corresponding user event otherwise an «even» user event

Log4Prolog: a tango LogViewer branch



This work has been made possible by the financial support of the Italian Ministry of University and Research (MIUR)