Automated Fault Identification
(STATUS REPORT)

Hashem Waly

Groupe LSFM,
Département d'informatique et de génie logiciel,
Université Laval, Québec, Canada

June 29, 2010
Montréal, Canada

Team @Laval

- Hashem Waly
  - Master student.
  - Main resource in the project.
- Aymen Ben Ali
  - Undergraduate student.
  - Working as a summer trainee.
- Pape Maleye Niang
  - Master student working on a separate project (IDS):
    - Contribute to discussions.

Agenda

- Work done since December
- Demo
- Near Future
- Future Work

Work done since December

- Continuous reviews of the state of the art.
- Learning and exploring Eclipse plug-in framework
  - Exchanges with Ericsson team (François Chouinard and William Bourque).
- Definition of a scenario specification language:
  - Using ANTLR parser generator
    - Implementation of a parser.
    - Implementation of the corresponding classes.
- Implementation of Eclipse plug-ins:
  - Editor.
  - Checker (GUI + Detection engine).
  - And more …
- The developed plug-in is connected with Ericsson's plugins through *TMFDataRequest* API.

Current Status

- The defined language is partially supported.
- For the part of the language that is supported:
  - Editor:
    - Completion, Syntax coloring and Errors handling.
  - GUI (*Aymen has contributed*):
    - Add/delete/modify scenarios/group of scenarios through the editor.
    - Display dynamically the progress of the engine.
    - Currently in development.
  - Checker
    - Currently in development.

Demo

Methodology

- Step1: Define the core langage
  - *Filter predicates*: The language is composed of atomic parts (predicates). The smallest predicate is the *filter* which filters a specific field in the event (pid, process name, etc). Users can specify several filters on the different fields of the event related using relational operators.
  - *Event filters*: Grouping filters into the *event filter*. Tagging the filter with an *id* so it can be referenced by other filters.
  - *Scenarios*: Combining event filters together into a *scenario*.
  - *Scenarios as abstract events*: Through the parameters of a scenario, it could be used as a type in other scenarios.
  - *Group of scenarios*: Combining scenarios into a *group of scenarios*.

- Step 2: Beyond the core language
  - *Consuming events*, *non-consuming events*.
  - Advanced attributes: These attributes (priority, etc.) contribute to the quality of information communicated to other component of the project as the System Health component.

- Step 3: Define the scenario actions
  - Combine scenarios (or group of scenarios) with *actions* to form *rules*. Theses actions will be launched automatically when the engine observes a match for a given scenario or group of scenarios.

Near Future

- Support full syntax of the language:
  - Update the ANTLR specification.
  - Enhance the checker functionality.
  - Update the Editor with a semantic checker (type checking): this work is part of the summer undergraduate student.
- Evaluate the expressiveness of the language through the definition of several kind of scenarios.

Future Work

- Together with Aymen, measure the performance and try to optimize the algorithms by:
  - Compiling scenarios to more efficient structure (*Aymen work*).
  - Parallelization.
- Document the language and the implemented plug-in.
- Refine the actions part of the language

  scenarios -> actions

  - Synchronize with *System Health Monitoring* team (using for instance the IDMEF format).
- Implement steps 2 & 3.