MaCware User Manual

Version 0.99

Real-Time Systems Group
University of Pennsylvania

Preface

This document presents the user interface of the monitoring and checking (MaC) toolset. It describes the process of putting together a monitoring application from a system to be monitored and a set of scripts that describe what should be monitored and how. The language of the scripts is beyond the scope of this manual. Consult the MaC language reference manual for syntax and semantics of the languages used in MaC. The rationale and the general philosophy for MaC, as well as a description of the components of MaC framework can be found in [2, 1]. The current version of MaC is targeted towards Java programs.

The manual is organized as follows: Section 1 gives an overview of MaC framework. It presents the components of the framework and the steps necessary to produce a monitoring application. Then, Section 3 describes the graphical user interface for MaC and its use to configure MaC for a sample application. Finally, Section 4 gives the command-line interface to all MaC components.

1 MaC overview

The structure of the MaC framework is demonstrated in Figure 1. The user specifies the requirements of the system in a formal language. Requirements are expressed in terms of high-level events and conditions. In addition, a *monitoring script* relates these events and conditions with low-level data manipulated by the system at run time. Based on the monitoring script, the

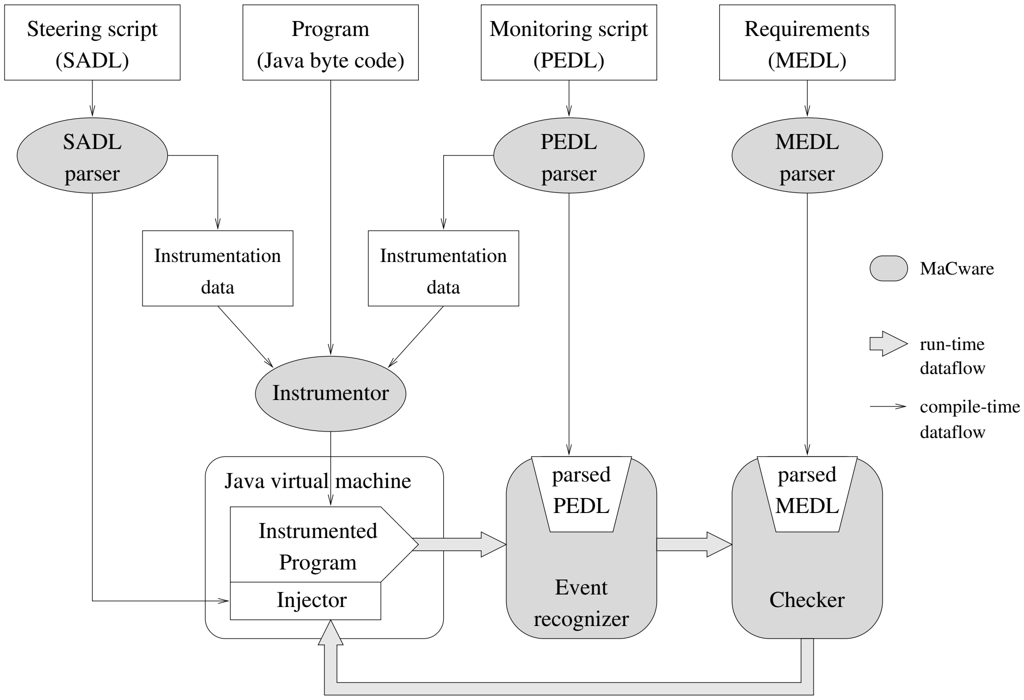

Figure 1: MaC architecture

system is *automatically* instrumented to deliver the monitored data to the *event recognizer*. The event recognizer, also generated from the monitoring script, transforms this low-level data into abstract events and delivers them to the run-time checker. The run-time checker is generated from the requirements specification. The run-time checker verifies the sequence of abstract events with respect to the requirements specification and detects violations of requirements.

The reason for keeping the monitoring script distinct from the requirements specification is to maintain a clean separation between the system itself, implemented in a certain way, and high-level system requirements, independent of a concrete implementation. Implementation-dependent event recognition insulates the requirement checker from the low-level details of the system implementation. This separation also allows us to perform monitoring of heterogeneous distributed systems. A separate event recognizer may be supplied for each module in such system. Each event recognizer may process the low-level data in a way specific to the respective module. For example, an event recognizer that is associated with a software component will work very differently from the one that processes traffic on a bus. But all event recognizers deliver high-level events to the checker in a uniform fashion.

In keeping with this design philosophy, two languages have been designed for use in the MaC framework. The Meta-Event Definition Language (MEDL) is used to express requirements. It is based on an extension of a linear-time temporal logic. It allows to express a large subset of safety properties of systems, including real-time properties. Monitoring scripts are expressed in the Primitive Event Definition Language (PEDL). PEDL describes primitive high-level events and conditions in terms of system objects. PEDL, therefore, is tied to the implementation language of the monitored system in the use of object names and types. MEDL is independent of the monitored system.

In addition, when steering is used, the user provides a steering script in addition to the monitoring script. The steering script describes steering actions and conditions for their invocation using the Steering Action Definition Language (SADL). Steering actions are invoked in response to requirement violations detected by the checker. The steering script generates additional instrumentation data for the system and also a special run-time component called *injector* that accepts action invocations from the monitor and triggers their execution within the system.

Thus, in order to set up monitoring of a system, the user has to perform the following steps:

1. Specify the monitoring script (PEDL) and compile it to generate the event recognizer.
2. Specify the requirements specification (MEDL) and compile it to generate the run-time checker.
3. Optionally, specify the steering script (SADL) and compile it to generate the injector.
4. Perform instrumentation of the system.
5. Start the runtime checker.
6. Start the event recognizer.
7. Start the system.

Note that the order in which MaC components are started, is important: the checker should be run before the event recognizer and both of them must be running before the system is started.

2 Installation of MaCware

MaCware [1] is written in Java and runs on version 1.1 or later version of java platform. MaCware uses JTREK library v1.1 for bytecode modification. Thus, user need to download JTREK v1.1 [2] and bug patch of JTREK v1.1. [3]. After downloading MaCware into <mac_dir> and JTREKv1.1 into <jtrek_dir>, user need to set CLASSPATH.

- For Unix user using tcsh, add

  ```
  setenv CLASSPATH $CLASSPATH":<mac_dir>:<jtrek_dir>:."
  ```

  at the last line of <home_dir>/.tcshrc. Then source <home_dir>/.tcshrc

[1] http://www.cis.upenn.edu/~rtg/mac/macware.zip

[2] http://www.digital.com/java/download/jtrek/index.html

[3] Patch is available at MaCware distribution (http://www.cis.upenn.edu/~rtg/mac/patch.zip). User should unzip patch.zip in <jtrek_dir>/dec/trek/

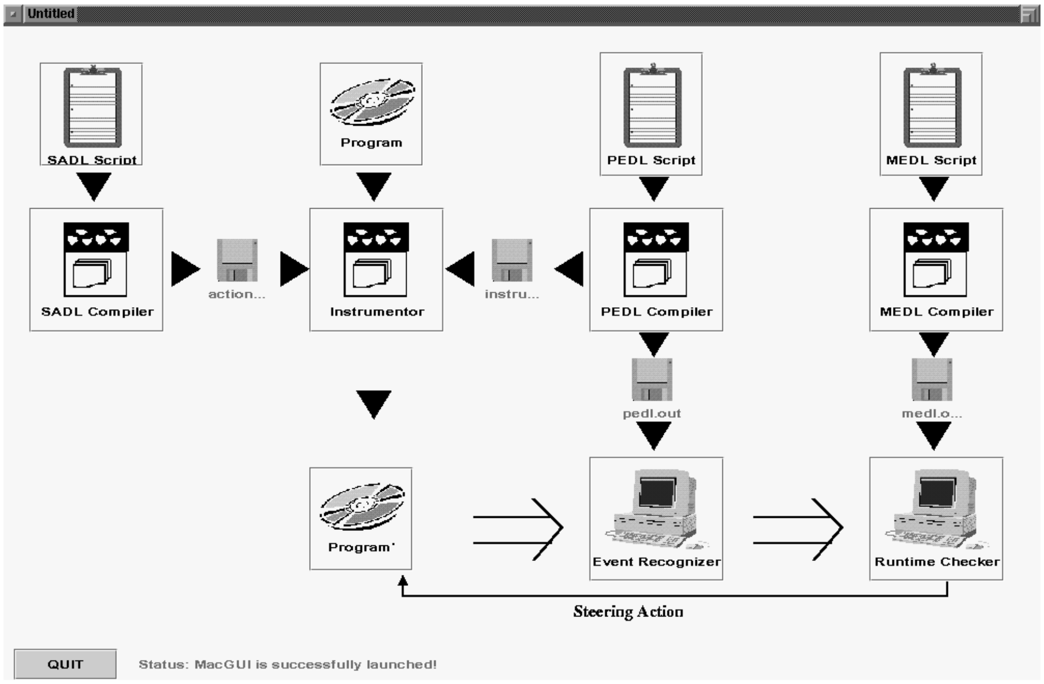

Figure 2: MaC control panel

- For Windows user, open `Start button` → `Setting` → `Control Panel` → `System` (→ `Advanced`) → `Environment` and add

  `;<mac_dir>;<jtrek_dir>;.`

  into the value field of the variable `CLASSPATH`

3 MaC GUI

The control panel for MaC is provided by the MacGUI application. It visually represents all MaC components that need to be set up in order to do monitoring, and guides the workflow by enforcing the dependencies between the components. For example, the checker cannot be set up before the user enters and compiles the requirements specification to be input to the checker.

Figure 2 shows the view of the MaC control panel. It has been intentionally made to follow the layout of Figure 1. Each component of the MaC framework is represented as a box in the control panel. Components that are ready to be configured are enabled. When the user selects an enabled component with the mouse, a dialog comes up that allows the user to configure it. When the configuration of the component is completed, other components that depend on it become enabled.

There are three script boxes that allow the user to edit the monitoring and steering scripts and the requirements specification. When the user selects a script box, an editor window comes up, where the user can enter the script or load it from a file. Alternatively, a script is specified when the corresponding compiler box is enabled, through a file dialog.

A dialog associated with the program box allows the user to specify a semicolon-separated list of class names that need to be instrumented. The instrumentation box is activated after the class list has been given and the monitoring script has been compiled. The dialog of the instrumentation box allows the user to specify the media for communication with the event recognizer and the checker. Although MaC interaction, in general, can be done either through TCP/IP among different hosts or via FIFOs, the control panel allows only TCP/IP communication on the same host where the control panel runs. Therefore, the user has to specify the port address of the event recognizer and, if steering is used, the port number to listen for steering action invocations is also required. This data is inserted by the instrumentor into the initialization code of the instrumented program.

The bottom row of boxes starts the program and the monitoring components. The dialogs for the event recognizer lets the user specify the port number of the checker and the port number to receive data from the program. The checker dialog accepts the port number for messages from the event recognizer and, in the case of steering, the address of the injector. Be sure that the port number used by the event recognizer and the runtime checker must not be used by other programs.

Starting the program, the user specifies the class file where the execution starts and gives the run-time parameters of the program. All monitoring-related information has already been specified during instrumentation.

Finally, the bottom line contains the quit button and shows the status of MaC activity.

Starting the control panel. The MaC control panel is a Java application that is started through class `mac.gui.guiLauncher.MacGUI` with no parameter. [4]

4 Command-line MaC interface

All MaC components can be started from command line. Command-line interface provides larger flexibility than the control panel. Below, we give the executable class name of every component and parameters that the component can accept, as well as the output produced by each component. All MaC components are Java appplications.

MEDL compiler. The executable class is

`mac.runtimeChecker.medlParser.MedlParser`

and can take a single argument, the file containing a script to compile. If no argument is given, the parser reads from standard input. The compiled script is stored in the file `medl.out`.

PEDL compiler. The executable class is

`mac.eventRecognizer.pedlParser.PedlParser`

and can take a single argument, the file containing a script to compile. If no argument is given, the parser reads from standard input. The compiled script is stored in the file `pedl.out`, which is used by the event recognizer. In addition, instrumentation data, used by the instrumentor, are stored in the file `instrumentation.out`.

SADL compiler. The executable class is

`mac.steering.parser.SadlParser`

and can take a single argument, the file containing a script to compile. If no argument is given, the parser reads from standard input. The compiled script is stored in the file `actions.out`. In addition, the compiler produces and stores in the current directory a Java source file named `foo_Injector.java`, where `foo` is the name of the steering script. The file must be compiled into byte code, which will be loaded into the JVM together with the monitored program.

[4] MacGUI requires Swing 1.1

Instrumentor. The instrumentor application is used when there is no steering involved. [5]

`mac.filter.instrumentor.Instrumentor`

and the following arguments are required in the order shown:

- the list of monitored objects, produced by the PEDL parser as `instrumentation.out`.
- a semicolon-separated list of class files to be instrumented. Class files are assumed to be in the current directory.
- The address of the event recognizer or the name of a file or FIFO where the monitored data will be output.

If steering is to be used, a different instrumentation tool is used, located in `mac.steering.SteeringInstrumentor`. Steering instrumentor allows to specify its arguments in any order with the following keys:

- `-monitor` denotes that the following argument is the monitoring instrumentation data, that is, the list of monitored objects.
- `-steer` denotes that the following argument is the steering instrumentation data, steering actions and positioning information.
- `-ports` denotes that the following two arguments are the address of the event recognizer and the port number for action invocations, in that order. When steering is used, the MaC run-time components have to communicate through TCP/IP.
- `-classfiles` denotes that the following argument is the list of classfiles to be instrumented.

Run-time checker. The executable class is

`mac.runtimeChecker.interpreter.CheckerMain`.

The first argument specifies the input stream for the checker. It is a port number on which to listen for connection requests from the event recognizer or, if preceded by the `-f` key, the file or pipe name from which event recognizer messages are read. The second, optional argument, denoted by `-s`

[5] The instrumentor generates a policy file `mac_policy` which allows communication from a target program to the event recognizer. See more detail in `http://java.sun.com/products/jdk/1.3/docs/guide/security/PolicyFiles.html`

key, denotes that steering will be used in this run and specifies the address of the system to be steered in the form `hostname:port`. The hostname can be omitted, in which case the system is assumed to be executed on the same machine as the checker. The checker then establishes a connection to the injector during start-up. The last argument is the compiled MEDL script produced by the MEDL compiler.

Event recognizer. The executable class is

`mac.eventRecognizer.interpreter.EventRecognizerMain`.

The following arguments are required in the order shown:

- the port number for the monitored data sent by the instrumented program, or a file name preceded by the `-f` key.
- the address of the checker (hostname may be omitted), or a file name preceded by the `-f` key.
- the compiled monitoring script (`pedl.out`).
- the list of monitored objects (`instrumentation.out`).

Instrumented program. The program is started in the same way as it would without monitoring. All data necessary for communication with MaC components have been given to the program during instrumentation.

References

[1] M. Kim, M. Viswanathan, H. Ben-Abdallah, S. Kannan, I. Lee, and O. Sokolsky. A framework for run-time correctness assurance of real-time systems. Technical Report MS-CIS-98-37, University of Pennsylvania, 1998.

[2] M. Kim, M. Viswanathan, H. Ben-Abdallah, S. Kannan, I. Lee, and O. Sokolsky. Formally specified monitoring of temporal properties. In *Proceedings of the European Conference on Real-Time Systems - ECRTS'99*, pages 114–121, June 1999.