University of Coimbra

CISUC

Centre for Informatics and Systems of the University of Coimbra

BICEP RESEARCH PROJECT

BENCHMARKING COMPLEX EVENT PROCESSING SYSTEMS

FINCOS FRAMEWORK

USER GUIDE V.1.1

Coimbra, 31 August 2008.

Table of Contents

1. Introduction

The FINCoS framework is a Java-based set of tools for load submission and performance measuring of Complex Event Processing systems. It provides a flexible and neutral approach for experimenting diverse CEP systems, where multiple datasets, queries, answer Validators, and engines can be easily attached, swapped, reconfigured and scaled.

The FINCoS framework was developed in the ambit of the BiCEP research project, carried at the University of Coimbra, with funding of the Marie Curie European Commission.

Components and Applications

Figure 1 illustrates the general structure of the FINCoS framework, which includes five main components:

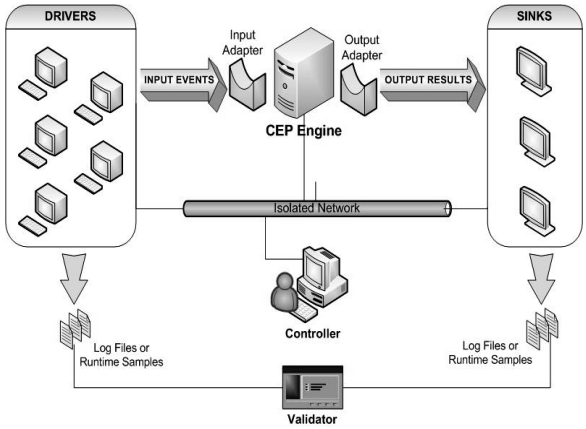


Figure 1.1: Architecture of the Framework

1. ***Driver*** – simulates external sources of events; it is responsible for submitting load to the system under test (SUT). The events which compose the workload can be generated by the Driver itself according to user's specification or they can be loaded from a third-party file. Currently, the FINCoS framework supports only simple directives for data generation, so it is likely that one needs to use its own dataset obtained directly from real applications or from simulations;
2. ***Sink*** – receives output events resulting from the queries running on CEP engines. The results are stored in log files and/or transmitted over network for subsequent validation;
3. ***Controller*** – it is the interface between the framework and the user. The Controller application is used to configure the setup of the environment (e.g., number of drivers and sinks or how many machines are used) and to control the other applications during performance tests (e.g., start and interrupt components, increase/decrease load intensity or change workload parameters);
4. ***Adapters*** – there is no standard event representation across CEP engines, so typically each product has its own set of supported formats. In the absence of a common format to all engines, we have adopted a neutral comma-separated-value (CSV) event representation and custom adapters to make the conversions to a format compatible with their corresponding CEP engine. We have implemented a few adapters for some products, and it should be easy for other people to extend that list – that is especially the case for some vendors whose products already support, fully or partially, event exchange using CSV format;

5. ***Validator*** – validates the results produced by CEP engines. It takes information from all drivers and all sinks and produces summaries indicating how well the SUT has performed. Those reports can be displayed while the tests are running or only after completion. A typical report includes performance metrics such as response time or throughput, as well as information about the correctness of the results. *Validators are query-specific and as such must be dynamically developed and attached to the framework*.

Tests can be configured to use multiple Drivers and Sinks distributed over different machines. This architecture permits to increase the load over the SUT by scaling up the number of components, when the current configuration is not powerful enough for submitting the desired load. The framework also provides flexible experiments setup. For instance, each Driver has its own workload (i.e., one or more datasets and event submission rates) and executes independently from other Drivers – which can be useful to simulate events coming from distinct sources. Of course, all Drivers can have exactly the same configuration (for instance, to increase the load over the SUT as discussed before). Moreover, the workload of a Driver can be made very dynamic, either during test setup, by dividing its execution in one or more sequential phases, with each phase having its individual workload characteristics (event rate, duration and dataset) or by altering some workload parameters on-the-fly, while tests are running. The possibility to vary workload over time is useful for testing the ability of CEP engines to adapt to changes. Finally, the framework was designed to be portable across different CEP products and test scenarios. The parts for which there is no standardization, namely, *adapters and Validators*, were made "plug-and-play" components, which *are developed and attached to the framework on demand*.

The FINCoS framework is currently distributed as package of four applications. Drivers and Sinks are encapsulated inside a single application called ***FINCoS daemon service***. This application should run in background in every machine where Drivers and/or Sinks are expected to run. Its job consists simply in starting new instances of Drivers and Sinks. The controller component is implemented as an application called ***FINCoS Controller***. The framework also includes a small application, called ***FINCoS Adapter***, for receiving events from Drivers and forwarding them to the CEP engine and for receiving events from CEP engines and forwarding them to Sinks. The vendor-specific conversions are performed by classes inside this application. These classes are developed on demand, according to the need for supporting a given CEP product (the framework currently includes such classes for two CEP engines). Finally, the FINCoS framework comes with a sample validation application, called ***FINCoS Performance Monitor***, which currently supports only runtime performance measuring. This application will be soon extended to incorporate correctness check and offline validation (using log files produced by Drivers and Sinks).

Troubleshooting

If you experience `OutOfMemory` errors with the FINCoS Daemon service application, try to increase the heap memory size available for it. For doing that, edit the" `–Xmx`" parameter in the corresponding batch file. The following line shows an example of a possible configuration for maximum and minimum heap size parameters:

```
-Xmx512m -Xms32m
```

In this case, the available memory is constrained to a range between 32MB and 512MB.

If you identify any problems or bugs with the FINCoS framework, please send an email to mnunes@dei.uc.pt , with the problem description.

2. Installation

Currently the FINCoS framework is delivered as a .zip file and some manual steps are still required before using it (we plan to create an installer soon to eliminate these steps). In order to install the FINCoS framework:

1. Make sure that a Java Virtual Machine is installed in any machine where any component of FINCoS is expected to run. (*FINCoS was tested with JRE's version 1.5 and 1.6*);
2. Add the Java home to Windows environment variables (right click on "My Computer" -> "Properties" -> "Advanced" –> "Environment Variables" -> "System Variables", edit the "Path" entry, by adding the JRE bin directory on its end);
3. Extract the FINCoS .zip file (on preference at the "C:/FINCoS" directory)
    a. If you extract the .zip file on a directory other than "C:/FINCoS" you will need to edit the RMI codebase property of the files "`run_Controller.bat`" and "`run_FINCoS_DaemonService.bat`" as follows

```
java.rmi.server.codebase=file: [instal_dir]/bin/
```

3. Creating a Test Setup

Tests are configured using the FINCoS Controller application. Creating a test setup consists in configuring at least one Driver and one Sink. Figure 2.1 shows the Controller application with a sample test setup.

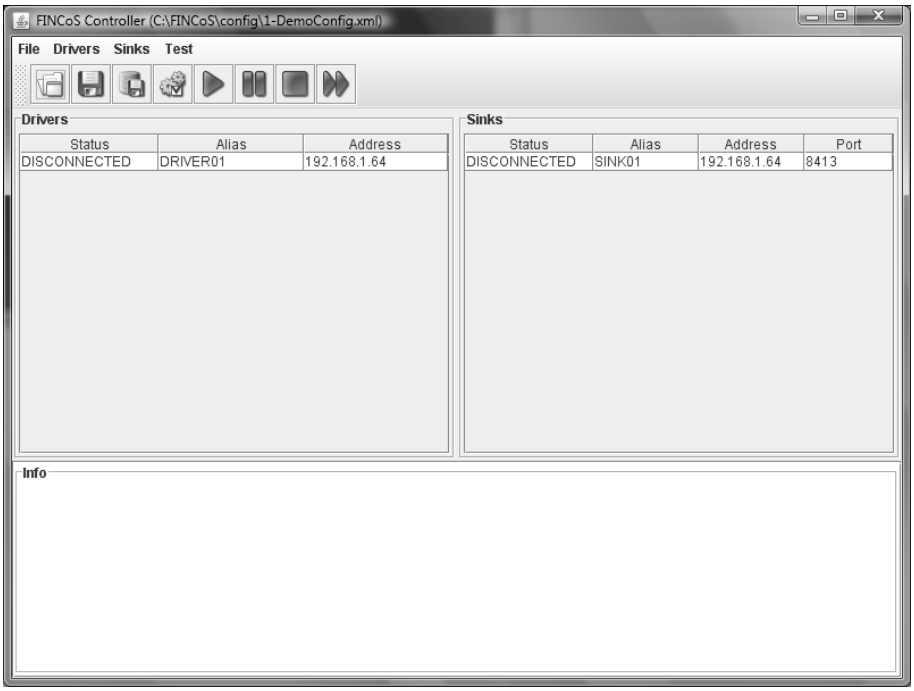


Figure 2.1: Controller Application showing a test setup with 1 Driver and 1 Sink

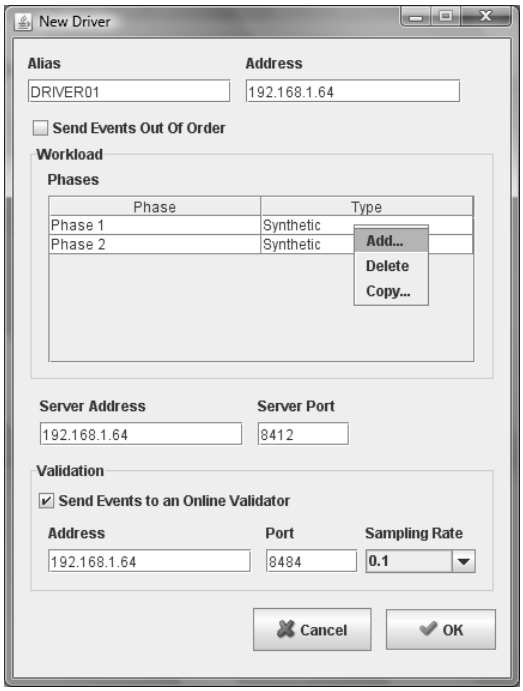
Tests setups can be saved in XML configuration files for being used again later. In next Sections we will see how to configure Drivers and Sinks.

Driver Configuration

To configure a new Driver, select 'New…' from the 'Drivers' menu. The "New Driver" Dialog will appear as in figure 2.2 (a). A Driver configuration has the following parameters:

- ✓ Alias – An unique identifier for the Driver in the test setup;
- ✓ Address – The IP address of the machine where the Driver must run;
- ✓ Sent events out-of-order – Indicates if events must be sent out of the order specified in the dataset;
- ✓ Workload – A set of phases, each with its own workload characteristics (we will talk more about workload parameters next). Each Driver must have at least one execution phase. You can add, delete and copy phases by right clicking the 'phases' table as shown in figure 2.2 (a);
- ✓ Server Address/Port – the address and port to which the Driver must send its events;
- ✓ Validation – Indicates if the Driver must send a percentage of its events ('sampling rate' parameter) to an Online Validator running at given machine on a given port (Validator 'address' and 'port' parameters). This parameters can be used, for instance, in conjunction with the FINCoS Performance Monitor application to provide real time performance information;

All the fields are mandatory and must be correctly filled in order to successfully finish the configuration of a Driver.

(a) 

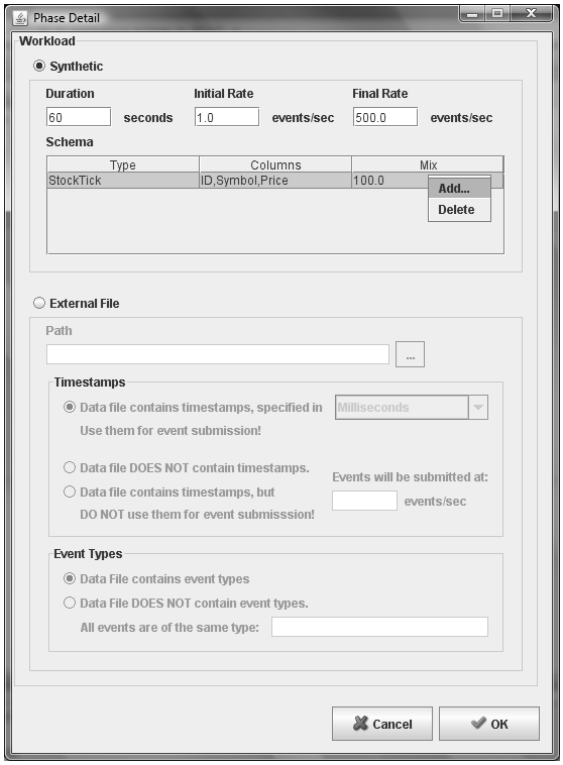
(b) 

Figure 2.2: (a) Driver Configuration Screen. (b) Workload Configuration Screen

Figure 2.2 (b) shows the configuration of the workload parameters of a Driver's phase. You can choose a synthetic workload to be generated by the FINCoS framework itself, or to load an external dataset file (the framework only supports external files stored in CSV – comma-separated-values – format).

For synthetic workloads, you will need to specify the desired duration of the phase, an initial event submission rate and a final submission rate (if the last two parameters are different, the event submission rate will increase/decrease linearly). Also, you will need to configure the schemas of the events to be generated (you must configure at least one event type). Figure 2.3 (a) shows the configuration of an event type named "StockTick", with three attributes: "ID", "Symbol" and "Price", of LONG, TEXT and FLOAT data types, respectively.

(a) 

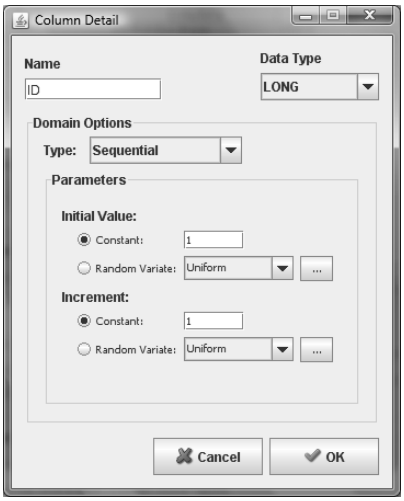
(b) 

Figure 2.3 (a) Configuring Event Type's Schema (b) Data generation Options

When creating event types, besides specifying their structures, you also configure the "domain" of each attribute, which controls the way data is generated. For instance, the "ID" attribute was configured as a sequential domain, with a fixed initial value of 1 and a fixed increment of 1 as shown in figure 2.3 (b). The framework currently supports the following data generation mechanisms:

Domain	Description
Random	Items are generated randomly, following a given variate distribution (uniform, normal or exponential).
Sequential	A domain that generates sequential data. An initial value and an increment are specified. Both the initial value and the increment can be constant values or random variates (uniform, normal or exponential).
Predefined List	The items are generated from a predefined list. It is possible to choose if the items from the list will be all generated in a predictable way, at the same proportion, or to specify an individual frequency for each item.

If you configure more than one event type, you can specify the percentage of events to be generated of each type by setting the 'mix' parameter in the schema table as shown in figure 2.2 (b) (the mixes do not need to sum up 100 or 1, because the framework normalizes the values; thus, specifying 50/50, 0.5/0.5 or 30/30 for two event types mixes will have the same effect: they will be generated in the same proportion)

For workloads based on external data files, you will need to inform the path of the dataset file as well as some information regarding timestamps and event types. Figure 2.4 shows the configuration of a workload with an external data file which contains timestamps but does not contain event types (all events are of the same type "Position_Report").

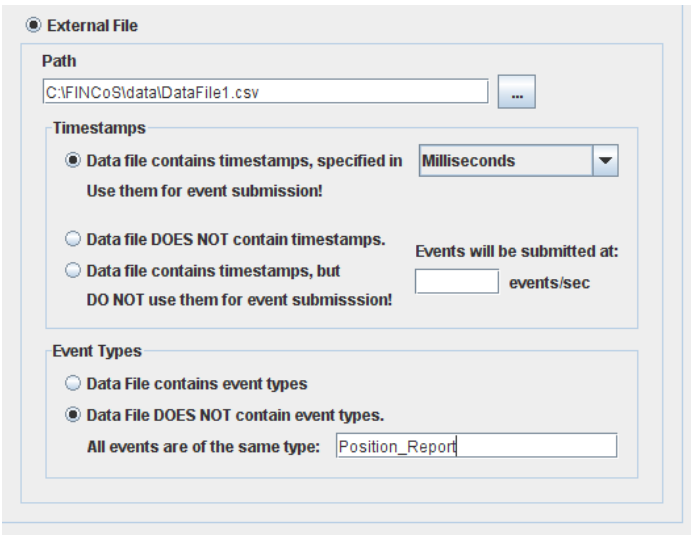


Figure 2.4 Configuration of a Workload with external dataset file

Regarding timestamps, there are three options:

i. ***The data file contains timestamps and they are used for event submission*** – in this case, the framework will send events according to the *relative* timestamps of the data file. The timestamp must be the first column in the CSV data file;

ii. ***The data file does not contain timestamps*** – in this case, you need to specify an event submission rate;

iii. ***The data file contains timestamps but they are not used for event submission*** – again, you must specify an event submission rate. It is important noticing that this is not the same as the latter case. If the data file contains timestamps and you select the second option, the framework will consider the timestamps as one more field of the events, and will send them to the CEP engine, which may not be the expected behavior. By selecting this third option, you are telling the framework to ignore the first column of the data file (the timestamp) and do not send it to the CEP engine.

Regarding event types, there are two options:

i. ***The data file contains event types*** – in this case, the framework will use the type information stored in the data file during event submission. The event type usually comes before the event's payload and must be preceded by a "type:" prefix;

ii. ***The data file does not contain event types*** – in this case, all events stored in the data file *must* be of the same type, and you need to specify the name of this type during workload configuration as shown in figure 2.4.

Sink Configuration

To configure a new Sink, select 'New…' from the 'Sinks' menu. The "New Sink" Dialog will appear as in figure 2.5. A Sink configuration has the following parameters:

- ✓ Alias – An unique identifier for the Sink in the test setup;
- ✓ Address – The IP address of the machine where the Sink will run;
- ✓ Port – The port on which Sink will receive events from the CEP engine;
- ✓ Streams – the list of output streams that the Sink will listen to. You can add and delete output streams by right clicking the 'Streams' list as shown in figure 2.5;
- ✓ Server Address – the address of the CEP engine/Adapter from which the Sink will receive its events. This information is not required for the Sink operation, but it is essential for the Adapter and Performance Monitor applications;
- ✓ Validation – likewise in the Drivers configuration, these fields indicate if the Sink must send a percentage of its events ('sampling rate' parameter) to an Online Validator running at given machine on a given port (Validator 'address' and 'port' parameters). In fact, it only makes sense configuring a Sink to send events to a Validator if the Drivers which generate the causer events of the output events received by the Sink are also configured to do so (validation usually involves correlating input and output events) and vice-versa;

Again, all fields are mandatory and must be correctly filled in order to successfully finish the Sink configuration.

Figure 2.5: Sink Configuration Screen.

---

* It is important noticing that the name of the output streams configured here must match the name of output streams in the CEP engine. Likewise, the name of event types specified during Drivers' configuration must match the name of input streams in the CEP engine.

4. Preparing the Environment for Running Tests

Before starting to run tests using the Controller application, a few steps need to be accomplished. First, the FINCoS daemon service application must be running in the machines where instances of Driver and Sink are expected to run, in order to start them. Second, the Adapter application must be running and ready for accepting connections from Drivers and for forwarding events from CEP engine to Sinks. Additionally, if Drivers and Sinks are configured to send part of their events to an Online Validator during runtime, this Validator must also be available before test starts (the FINCoS Performance Monitor can be seen as a "performance-only" Online Validator; you can use it to receive events from Drivers and Sinks and see performance metrics, such as response time and throughput in real time).

Starting the FINCoS daemon service application is very simple: you only need to start its process, by double-clicking its batch file (or the corresponding shortcut). Alternatively, you can configure this application to start on Windows startup. Next we see how to initialize the Adapter application.

Initializing the Adapter Application

The Adapter application receives events from Drivers as textual CSV messages, using plain sockets on a given local port. The message must contain event's data as payload and an additional property indicating event's type. The latter is used for routing purposes, to allow the Adapter to forward the event to the appropriate input stream on the CEP engine, after having done the required conversions (the event type's name which is specified in the message **must** match the name of an input stream in the CEP engine). Likewise, resulting events from the CEP engine are delivered using plain sockets on a remote port of appropriate Sinks. Notice that somehow the Adapter needs to associate output streams on CEP engine to Sinks, in order to deliver resulting events to the right destination. That's why the Adapter application needs to be initialized before running tests. The following picture shows the initial state of the Adapter application, just after being started:

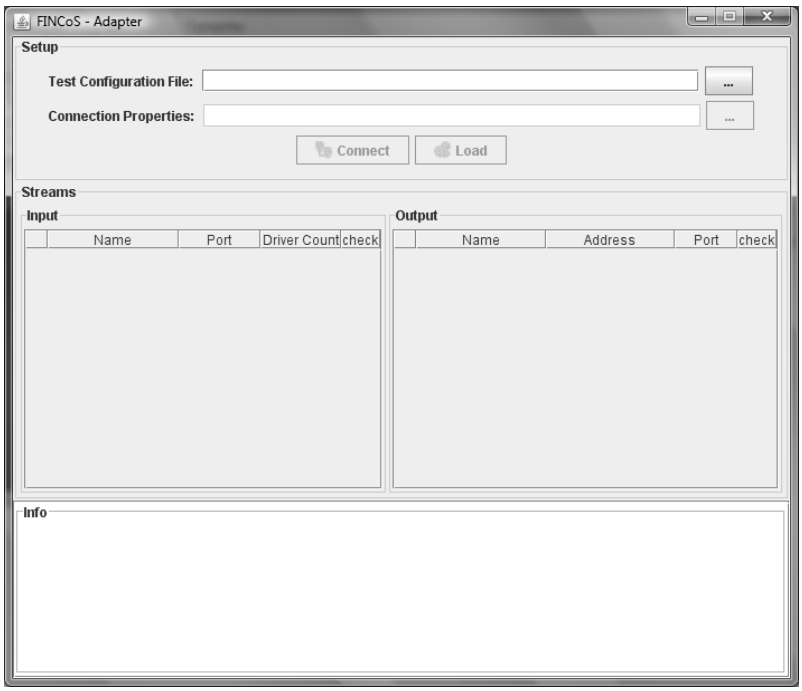


Figure 3.1: Adapter application immediately after being started

As we can see, no information about the streams is available at that moment. The initialization of the Adapter is then done as follows:

1. ***Load a test configuration file*** – in this step, you must inform the path of a configuration file, previously created with the Controller application, containing the setup of a test. At that moment, the Adapter will get and display the list of input and output streams which are intended to be used in the test, as shown in figure 3.2;
2. ***Load a file containing vendor-specific connection properties*** – This file contains information like the IP address of the machine where the CEP engine is running, the TCP port for connection, and any other vendor-specific parameter needed for allowing the Adapter application to connect with the intended CEP engine. Of course, different CEP products will have different fields in this .properties file. If you want to extend the Adapter application for supporting new CEP products you will need to create your own .properties file and write some code to parse it;
3. ***Connect to CEP engine*** – when the user clicks on the "Connect" button, the Adapter will try to connect with the CEP engine using the connection properties obtained from the vendor-specific .properties file (in most CEP products the continuous queries must already be running at that moment). If the connection succeeds, the Adapter will retrieve the list of available input and output streams. The Adapter then will check if the streams configured in the test setup file are all available in the CEP engine. If that is the case, the Adapter application will look like in figure 3.3, with a check mark in the right side of each stream, indicating that they are indeed available at the CEP engine. At that point, the Adapter will be ready for being loaded. On the other hand, if some of the streams in the test setup file are not present in the CEP engine, the Adapter will display a warning message, and it will not be possible to load it (in fact, it is still possible to proceed with driver load in this case, by deselecting the missing stream(s); however the missing stream(s) will not be available for use at the Adapter, which may lead to an undesirable test behavior);
4. ***Load the Adapter*** – when the user clicks the "Load" button, the Adapter application initializes its communication interfaces, by opening server sockets for receiving events from Drivers and by subscribing to output streams at the CEP engine. At that time, the Adapter is fully initialized and can already be used in performance tests;

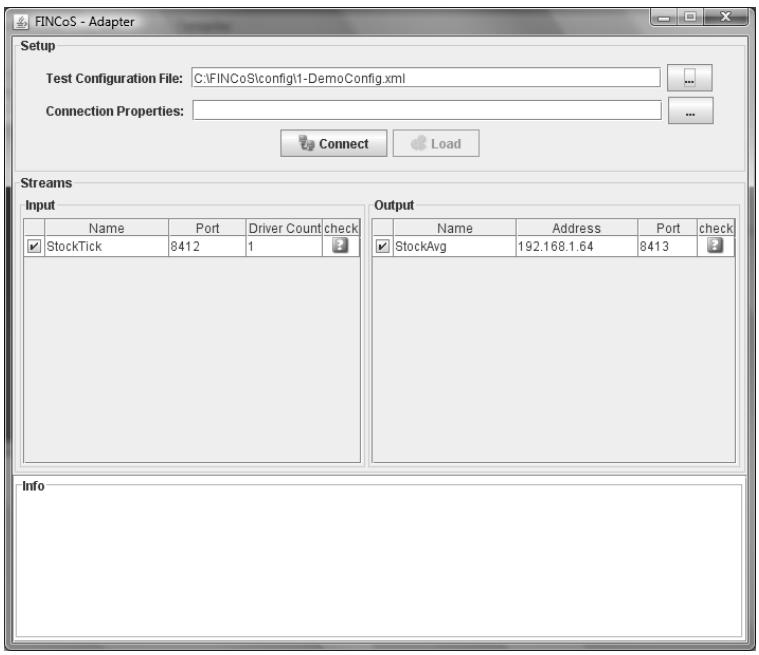


Figure 3.2: Adapter application after opening test setup file

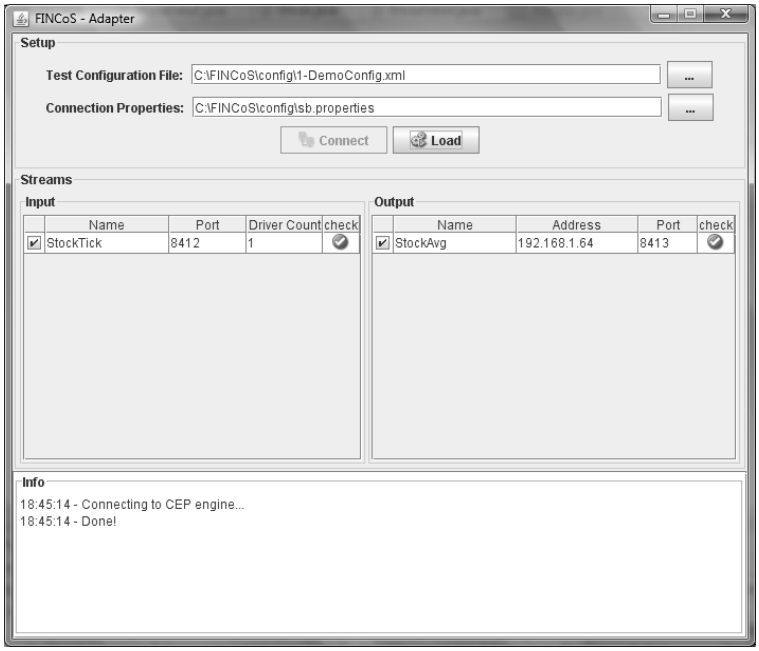
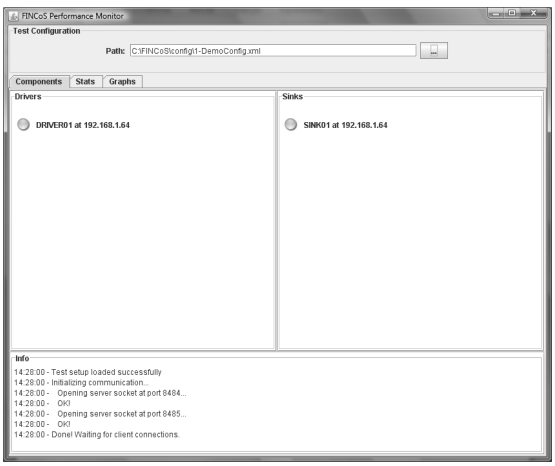


Figure 3.3: Adapter application after connecting to CEP engine

Initializing the Performance Monitor Application

The FINCoS Performance Monitor application permits to see the response time and throughput of each query being exercised during a performance test. Initializing it consists simply in informing the path of the test configuration file. This allows the FINCoS Performance Monitor application to know which queries it is expected to monitor and to identify exactly from which component (Drivers and Sinks) the events are coming. Figure 3.4 (a) shows a test setup where one Driver and one Sink were configured to send events to the FINCoS Performance Monitor application (this information is retrieved by a Performance Monitor instance by comparing the '*Validator address*' field of all Drivers and Sinks which are inside the test configuration file with the IP address of the machine where it is currently running). Figure 3.4 (b) shows the response time and throughput for two continuous queries.

(a)



(b)

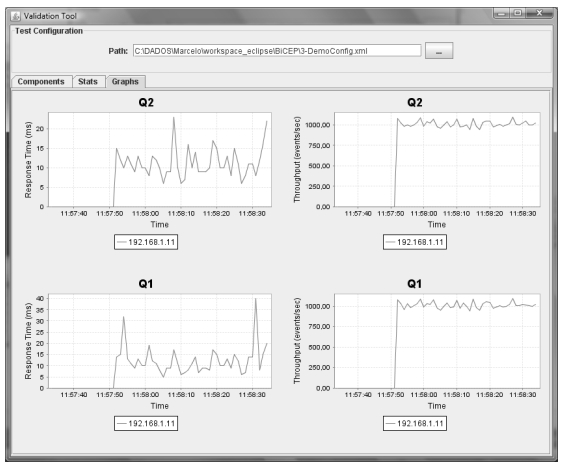


Figure 3.4 (a) The components from which the Performance Monitor will receive events
(b) Performance indicators for two continuous queries.

5. Running Tests

Having created a test setup, and prepared the environment, running the test is a very simple task and involves two actions (again, the auxiliary applications – namely the FINCoS daemon service, the FINCoS Adapter, and possibly the FINCoS Performance Monitor – must be running at that time):

1. ***Load components*** – by clicking the '*load*' button in the Controller application, all Drivers and Sinks will be initialized. For Drivers, that means connecting to the Adapter application and generating the dataset (or loading it from an external file, if it is the case). For Sinks, that means only opening a Server socket for accepting output events from the Adapter application. Additionally, if a Validator has been specified, Drivers and Sinks will try to connect with it too.
2. ***Start load submission*** – by clicking the '*start*' button in the Controller application, all Drivers will begin to send events to the Adapter, which in turn will forward them to the CEP engine.

** It is possible to load or start a component individually by right clicking it in the 'Drivers' and/or 'Sinks' table.*

Changing Parameters in runtime

The FINCoS framework allows changing some workload parameters while test is running. For instance, it is possible to alter the rate at which events are submitted to the CEP engine, by specifying a multiplication factor. The original event submission rate will then be multiplied by the specified factor. Notice that these modifications are not cumulative; that is, by setting the multiplication factor to 2 twice does not make the events to be submitted 4 times faster, but 2 times faster).

It is also possible to change datasets by switching to the next phase of a Driver execution (of course, this requires that different phases with different datasets have been previously configured). All these modifications can be applied to only a single Driver or to all Drivers at once.

Pausing/Stopping Load Submission

At any moment it is possible to pause and stop load submission. Pausing load submission means that all Drivers will temporarily interrupt event submission, but nothing happens to the Sinks. Stopping the load submission interrupts definitely event submission at Drivers and unloads them (closing the connection with the Adapter). The Sinks are also unloaded (closing their server socket, and thus, the connection of the Adapter application to them).

Performance Test Workflow

In this Section we review the whole flow of steps needed for running a test with the FINCoS framework:

1. [*Controller, user*] ***Start the Controller application and use it to create a new test setup or load a previously saved configuration file***;
2. [*Manually or automatically by OS*] ***Start the FINCoS daemon service in each machine where Drivers and/or Sinks are intended to run***;
3. [*Manually*] ***Start and Load one or more instances of the FINCoS Adapter Application***;
   - Description:
     - ✓ [*Adapter, user*] Informs paths for test configuration file and connection properties file;
     - ✓ [*Adapter, application*] Connects to CEP engine and checks stream compatibility;
     - ✓ [*Adapter, application*] Initializes server socket for Drivers and output listeners for Sinks;

4. [*Manually*] ***Start and Load one or more instances of the FINCoS Performance Monitor Application***;
   - Description:
     - ✓ [*Perfmon, user*] Informs path for test configuration file;
     - ✓ [*Perfmon, application*] Initializes server sockets for Drivers and Sinks;
5. [*Controller, user*] ***Load all Drivers and Sinks***;
   - Description:
     - ✓ [*Controller, user*] Clicks 'load' button;
     - ✓ [*Controller, application*] Transmits the corresponding configuration for each Driver and Sink;
     - ✓ [*Driver, application*] Generates/Loads data files and connects to Adapter application;
     - ✓ [*Sink, application*] Initializes server socket to receive output events from Adapter application;
6. [*Controller, user*] ***Start load submission***
   - Description:
     - ✓ [*Controller, user*] Clicks 'start' button;
     - ✓ [*Controller, application*] Sends a remote command for Drivers to start load submission;
     - ✓ [*Driver, application*] Submits events previously generated/loaded;
     - ✓ [*Adapter, application*] Receives incoming events from Drivers, makes the appropriate conversions, and forwards to CEP engine; receives output events from CEP engine, makes the appropriate conversions, and forwards to Sink;
     - ✓ [*Sink, application*] Processes output events from Adapter;

7. Pause load submission

- Description:
  - ✓ [*Controller, user*] Clicks 'pause' button;
  - ✓ [*Controller, application*] Sends a remote command for Drivers to pause load submission;
  - ✓ [*Driver, application*] Interrupt temporarily event submission;

8. Stop load submission

- Description:
  - ✓ [*Controller, user*] Sends a remote command for Drivers to stop load submission and unload them as well commands for unloading Sinks;
  - ✓ [*Driver, application*] Interrupts definitely event submission and disconnect from adapter;
  - ✓ [*Sink, application*] Processes any pending events in the server socket, and then closes it;
  - ✓ [*Adapter, user*] Reloads Adapter or closes application;

6. Adding support for new CEP engines

Currently the FINCoS framework offers adapters for only two CEP engines, namely Coral8™ and StreamBase™. However, it must be easy extending this list, since developing an adapter in the FINCoS framework basically consists in implementing two functions: one to send events to the CEP engine and another to receive events from it. The conversion between CSV records and product's internal representation is encapsulated inside those two functions. The following code snippet shows the abstract class `CEPEngineInterface` used by the FINCoS Adapter application to connect to CEP engines, send and receive events to/from them, and retrieve list of streams:

```
public abstract class CEPEngineInterface {
	/**
	 * Connects to CEP engine
	 */
	public abstract boolean connect() throws Exception;

	/**
	 * Performs any vendor-specific initialization at client side
	 * (e.g. initialize listeners for output streams)
	 */
	public abstract boolean load() throws Exception;

	/**
	 * Disconnects from CEP engine and performs any finalization
	 * procedures needed (e.g. close listeners for output streams).
	 */
	public abstract void disconnect();

	/**
	 * Sends an event to an input stream at CEP engine
	 * Converts the framework CSV-based representation to a format
	 * supported by the CEP engine.
	 */
	public abstract void sendEvent(String e) throws Exception;

	/**
	 * Optional function. Retrieves the list of input streams of a
	 * continuous query running on CEP engine.
	 */
	public abstract String[] getInputStreamList() throws Exception;

	/**
	 * Optional function. Retrieves the list of output streams of a
	 * continuous query running on CEP engine
	 */
	public abstract String[] getOutputStreamList() throws Exception;
}
```

For extending the Adapter application in order to support other CEP engines, you need to create a class that inherits from the `CEPEngineInterface` class and implement its abstract methods. You will also need to implement a listener for receiving events from output streams at the CEP engine. You can find more details by looking at the FINCoS framework source code.