A framework for development of concurrency and I/O in servers

Development of concurrency and I/O in servers and middlewares becomes more and more complex:
- minimization of latency;
- maximization of bandwidth;
- no consensus on the best concurrency model;
- select the model best adapted to the hardware.

Applications are modeled by a directed graph, in which each stage (or vertex) corresponds to an atomic unit of treatment and edges correspond to channels (method calls, local queues or sockets) between them.

We describe here the implementation of a simple “Echo” server which uses three stages. The directed graph models the interconnection of its stages:

```
accept -> read -> write
```

Specifications and code generation are 100% Java! This ensures the portability of the applications developed using our framework.

### Development process

This table summarizes the development steps of our framework:

<table>
<thead>
<tr>
<th>Input / Output interfaces</th>
<th>specified in Java by user</th>
<th>Events</th>
<th>generated from interfaces</th>
<th>Functional code of a stage</th>
<th>specified in Java by user</th>
<th>Technical code of a stage</th>
<th>generated from concurrency</th>
<th>Concurrency selection</th>
<th>specified by user</th>
<th>Concurrency</th>
<th>generated from concurrency</th>
</tr>
</thead>
</table>

### Stage description

The developer should implement the `handle(...)` method which corresponds to the instructions carried out by a stage. Its parameters are the input and/or output events and the context.

The context is the way to reach successor(s) in the graph.

**Example:**

```
public class AcceptStage {
  public void handle(StageContext ctx, OutputAcceptEvent out) {
    SaburoServerSocket server = out.getSaburoServerSocket();
    ctx.dispatchToSuccessor(out);
  }
}
```

### Communication generation

The interfaces previously defined of the input and/or output events which allow the communication between stages are automatically generated.

The implementation of the context is also automatically generated according to the concurrency model.

### Concurrency generation

The last step consists in the automatic generation of the concurrency model.

**Example:**

**Iterative architecture**

```
public class IterativeModel {
  public void service() throws Exception {
    new Thread(new Runnable()) {
      public void run() {
        while (true) {
          writeSelector.doSelect();
          if (writeSelector.select()) {
            new Thread(new Runnable()) {
              public void run() {
                while (true) {
                  readSelector.doSelect();
                  if (readSelector.select()) {
                    acceptSelector.doSelect();
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

**Example:**

**Staged Event-Driven Architecture**

```
public class SedaModel {
  public void service() throws Exception {
    new Thread(new Runnable()) {
      public void run() {
        while (true) {
          writeSelector.doSelect();
          acceptSelector.doSelect();
        }
      }
    }
  }
}
```

The bytecode is generated automatically using ASM and all the code generators can be used at runtime, even if they are usually used at compile time.

### Event description

The developer has to define the interface for input and/or output events for each stage. These events allow the communication between stages.

**Example:**

For the initial stage, only an output interface is defined:

```
public interface OutputAcceptEvent {
  public void setAcceptSaburoSocket(SaburoSocket s);
}
```

For a final stage only an input interface is defined:

```
public interface InputWriteEvent {
  public SaburoSocket getAcceptSaburoSocket();
  public byte[] getReadByteBuffer();
}
```

For any other stage input and output interfaces should be defined:

```
public interface InputEvent {
  public SaburoSocket getAcceptSaburoSocket();
  public void setAcceptSaburoSocket(SaburoSocket s);
  public byte[] getReadByteBuffer();
  public void setReadByteBuffer(byte[] b);
}
```

### Stage connections

The connection of the stages has to be specified in Java by the developer.

**Example:**

```
StageManagerImpl manager = new StageManagerImpl();
manager.connect(AcceptStage.class, ReadStage.class);
manager.connect(ReadStage.class, WriteStage.class);
```

### Concurrency selection

The concurrency model has to be selected in Java by the developer.

**Example:**

```
ModelExecutorImpl executor = new ModelExecutorImpl();
evaluator.run(configurator, stageManager, SEDA);
```

Currently, these two steps are hand-coded but could be generated automatically via an Eclipse plugin.