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

To cite this version:
Gautier Loyauté. A framework for development of concurrency and I/O in servers. 1st European Conference on Systems (EuroSys 2006), Apr 2006, Belgium. 1pp., 2006. hal-00620056
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>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>generated from interface</td>
</tr>
<tr>
<td>Functional code of a stage</td>
<td>specified in Java by user</td>
</tr>
<tr>
<td>Technical code of a stage</td>
<td>generated from concurrency</td>
</tr>
<tr>
<td>Stage connection</td>
<td>specified by user</td>
</tr>
<tr>
<td>Concurrency selection</td>
<td>generated from concurrency</td>
</tr>
</tbody>
</table>

### 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:

```java
public interface OutputAcceptEvent {  
    public SaburoSocket getAcceptSaburoSocket();
}
```

For a final stage only an input interface is defined:

```java
public interface InputWriteEvent {  
    public SaburoSocket getAcceptSaburoSocket();
    public ByteByteBuffer getReadByteBuffer();
}
```

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

```java
public interface InputReadEvent {  
    public SaburoSocket getAcceptSaburoSocket();
    public ByteByteBuffer getReadByteBuffer();
}
```

### 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:**

```java
public class AcceptStage {  
    private final SaburoServerSocket server;
    public void handle(stageContext ctx,  
        OutputAcceptEvent out) {
        SaburoSocket client = server.accept();
        out.setAcceptSaburoSocket(client);
        ctx.dispatchSuccessor(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.

**Example:**

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

### Concurrency selection

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.

**Example:**

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

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

---

Gautier Loyauté

Université de Marne la Vallée
Laboratoire d’Informatique de l’Institut Gaspard-Monge
UMR-CNRS 8049
F-77454 Marne la Vallée, Cedex 2, France
e-mail: loyaute@univ-mlv.fr