Development of a Distributed System Applied to Teaching and Learning

Erik Ramos
Universidad Tecnológica de la Mixteca
Oaxaca, México

September 5, 2009
Table of contents

1. Introduction
2. Teaching and learning
3. Methodology
4. Architecture
5. Internal Level
This presentation is about how erlang helps us to build a Teaching and Learning application.
Introduction

In real world are a lot of parallel activities

1. House
2. School
3. Office Work
In classrooms are many concurrent tasks
Students meet for do homework
Teachers meet for do a test

I think ...

The question 1 must be ...

I can't believe
The system complements the traditional classroom environment (support some of the concurrent task)
Why??

- Our customers are our teachers and students
- We can help us with our knowledge and experience
- Rapid prototyping
- We can share ideas in every stage of implementation
- We write only necessary documentation
Declarative Programming

Why??

- Erlang code is close to specification
- Is a good complement to formulate rapid prototyping
A centralized distributed System
server is a process in execution from a node
client is a process requiring the services or resources from the server
Architecture

Diagram showing a three-way communication structure with nodes labeled $C_1$, $S$, and $C_2$. The diagram illustrates a distributed system with nodes $C_1$ and $C_2$ connected to a central node $S$. This structure is part of the development of a distributed system applied to teaching and learning.
Development of a Distributed System Applied to Teaching and Learning

Erik Ramos
Universidad Tecnológica de la Mixteca
Oaxaca, México

Architecture

Diagram:

- GS
- GC1
- C1
- GC2
- C2
Internal Level

- Useful to characterize the internal flow of actions
- We use CCS Calculus of Communicating System
- We model three special processes: Admin, Student and Teacher
- This approach takes into account the Erlang capacity to generate processes (message-passing)
Internal Level

- Admin: coordinates the overall system by allowing or blocking other processes
- Student and Teacher interact with a human
Internal Level

```
Student def Student.Login

Login def
    ReqAdminAut(user, pswd).(Ok +
    Bad +
    NoMoreAttempts +
    NoGoodClosing.Recover +
    cancel.Student)

Recover def
    LoadStudentLastGoodState.startEnv(stateInfo)

Ok def
    goodLogin.LoadStudentPreferences.StartMenu +
    goodLogin.LoadStudentLastActivity.startEnv(lastActivity)

Bad def
    timerOut.LockStudentAccount +
    wrongUser.increaseCounter.Login +
```
To specify the system behavior from the external point of view, we use use-case model of the UML. This model represents the interaction among distinct actors (students, teachers, or admin). With use-case, we can identify unusual behavior of the system and give an answer to this behavior.
Internal Level

Use Case: User login.
Brief Description: User proceeds to log into the system.
Scope: System.
Level: User goal.
Preconditions: User must previously be registered (through the subscribe command).
Post-conditions: User is successfully authenticated and log into the system.
Main Successful Scenario:
1. User introduces data for authentication by the system.
2. System notifies the user that he or she has been accepted.
3. System loads user preferences.
Extensions:
2a. System detects that the authentication is incorrect:
   2a.1 System notifies to the user that has been rejected.
   2a.2 System logs the event.
Designing test

- questionSA(question, answer, levelbloom)
- questionTF(question, answer, levelbloom)
- questionMO(question, answer, dis1, dis2, dis3, levelbloom)
Designing test

question(Question,optionsN(Options)),sol(Solution).

testTemplateServer() – >

testExample(comics,

[  
  question("Which is the color of Homer Simpson?",  
    option3("Yellow","Green","Blue"),sol(1)),  
  question("What animal is Donald?") ,  
    option4("Mouse","Duck","Dog","Pig"),sol(2)),  
  question("What is the favourite meal of Bugs Bunny?") ,  
    option2("Flowers","Carriots"),sol(2)),  
]).