Universidade do Minho
Departamento de Informática

Engenharia de Linguagens

Projecto Integrado

Software para Análise e Avaliação de Programas

Grupo 2:

José Pedro Silva
Mário Ulisses Costa
Pedro Faria

Braga, November 29, 2010

Abstract

resumo

Contents

List of Figures

List of Tables

Chapter 1

Introdução

Este relatório descreve o projecto desenvolvido para o módulo Projecto Integrado da UCE de Engenharia Linguagens do Mestrado em Engenharia Informática da Universidade do Minho.

Pretende-se que este projecto seja um *Software* para Análise e Avaliação de Programas (SAAP), tendo este *software* como objectivo criar um ambiente de trabalho, com um interface Web, que permita a docentes/alunos avaliar/submeter programas automaticamente.

1.1 Motivação

Motivação...

1.2 Objectivos

Este projecto tem como objectivos consolidar conhecimentos adquiridos nos diferentes módulos da UCE de Engenharia de Linguagens. Para isso, o *software* pretendido

1.3 Estrutura do Relatório

Estrutura do Relatório...

Chapter 2

Chapter2

Contextualização do cap 2...

2.1 Sec2

Sec 2

Chapter 3

Modelação do Problema

Contextualização do cap 2...

3.1 Modelação Informal

Com o diagrama da arquitectura do sistema, figura 3.1,pretende-se mostrar as várias entidades que podem aceder ao sistema, assim como as várias actividades que cada uma pode realizar e tarefas para o sistema processar. Também é realçada a ideia de que alguns dos recursos do sistema só estão dispóniveis ao utilizador depois de passar por outros passos, ou seja, o diagrama dá a entender a ordem pelas quais o utilizador e o sistema podem/devem executar as tarefas.

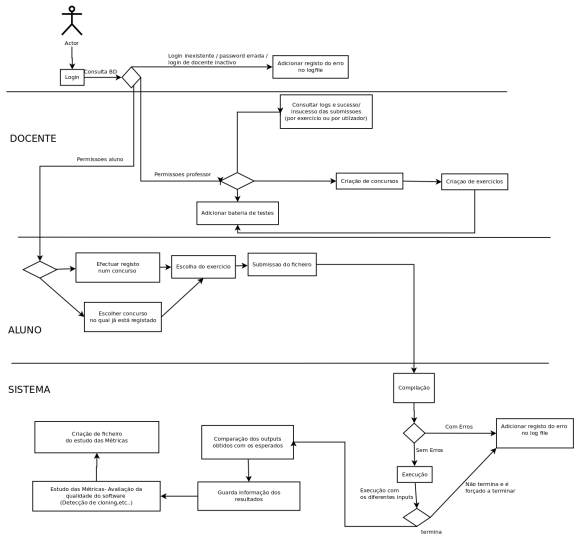

Figure 3.1: Diagrama de actividades::notanotanota

Para começar, como temos duas entidades diferentes que podem aceder ao sistema (o docente e o aluno/concorrente), dividiu-se o diagrama em duas partes distintas (uma para cada entidade referida), de modo a facilitar a leitura.

Em ambos os casos, o login é a primeira actividade que pode ser realizada. Se o login não foi efectuado com sucesso, é adicionado no log file uma entrada com a descrição do erro. No caso de o login ser efectuado com sucesso, consoante as permissões do utilizador em questão, tem diferentes opções ao seu dispôr.

No caso do login pertencer a um docente, este terá acesso aos dados de cada um dos grupos, podendo verificar os resultados que estes obtiveram na resolução das questões do(s) concurso(s), assim como ao ficheiro que contém a análise das métricas dos vários programas submetidos pelos mesmos. Poderá também criar novos concursos e os seus respectivos exercícios, assim como adicionar baterias de testes para os novos exercícios, ou para exercícios já existentes.

No caso do login pertencer a um aluno/concorrente, o utilizador terá a opção de se registar num concurso ou de seleccionar um no qual já esteja registado. Já depois de seleccionar o concurso, pode ainda escolher o exercício que pretende submeter. Depois de submeter o código fonte do programa correspondente ao exercício escolhido, e já sem a interacção do utilizador, o sistema compilará e tentará executar os diferentes inputs da bateria de testes do exercício, e compararar os resultados obtidos com os esperados. No fim de cada um destes procedimentos, serão guardados os resultados / erros. Para terminar, será feito um estudo das métricas do ficheiro submetido, tendo como resultado a criação um ficheiro com os dados relativos a essa avaliação.

3.2 Modelação Formal

Contextualização do cap 2...

3.3 Modelação à lá cenas

Formalização funcional do sistema.(...)

Listing 3.1: Part of grammar's definition code

```
// {pre condição}
//assinatura :: assinatura
//{pós condição}
//=~ (instancia de)

{ existsInDatabase(u) }
login :: (u =~ Username, Hash) -> SessionID -> Either Error SessionID
{ }

-----funçoes da pagina homePageProf ---
:: assinatura
data Dict a b = [(a,b)]
data Exercicio = Exercicio Enunciado (Dict Input Output)
data Contest =  Contest Nome Tipo [Exercicio]

{ existeSession(s)  and isProf(s) and (notEmpty . getExercice) c}
createContest :: s =~ SessionID -> c =~ Contest -> ()
{ (notEmpty . getDict) c }

{ existeSession(s)  and isProf(s) and (not . exist) (Exercicio e d)}
createExercice :: s=~ SessionID -> e =~ Enunciado -> d =~ (Dict a b) -> ()
{ exerciceCreated(Exercicio e d) }

{}
addTeste ::
{}

{ existeSession(s) and isProf(s) and contestIsClosed(c) }
consultarLogsContest :: s =~ SessionID -> c =~ Contest -> LogsContest
{ }

------funcoes da pagina homePageAluno -----

```

```
{ existeSession(s) and contestNotFull(c)}
efectuaRegistonoConcurso :: s =~ SessionID -> c=~ ContestName -> Credenciais
{ }

{ existeSession(s) and exerciceExist(e) }
submitExercicio :: s =~ SessionID -> e =~ Exercicio -> res =~ Resolucao ->
    rep =~ Report
{ rep = geraReport s e res }

{ existeSession(s) and exerciceExist(e) }
escolheExercicio :: s =~ SessionID -> e =~ Exercicio -> ()
{ }

{ existSession(s) and existContest(c) and userRegistadoNoContest(s,c) }
escolheConcursoJaRegistado :: s =~ SessionID -> c =~ ContestName -> ()
{ }

{ }
geraReport :: e =~ Exercicio -> res =~ Resolucao -> rep =~ Report
--{ rep = compile res >>= \p -> compare(e (execute p e)) >>= pageCerto) }
{ rep = do
             case compile(res) of
                 (Left error) -> geraReportBugCompile error res
                 (Right p) -> let resProps = execute p e
                                  in case (compare e resProps) of
                                      (Left certo) -> geraReportNoBug e res
                                      (Right errado) -> geraReportBugCompare
                                          errado res
}

geraReportBugCompile :: Exercicio -> Error -> Report
geraReportBugCompare :: Exercicio -> Errado -> Report
geraReportNoBug :: Exercicio -> Resolucao -> Report

{ }
compile :: Resolucao -> Either Error Program
{ }

-- Output do programa proposto
data ResolucaoProposta = Dict Input Output

{ }
execute :: Program -> Exercicio -> ResolucaoProposta
{ }

{ length(Exercicio)==length(ResolucaoProposta)}
compare :: Exercicio -> ResolucaoProposta -> Either Certo Errado
{ }

{ existSession(s) and existContest(c) and }
geraFinalReport :: s =~ SessionID -> c =~ Contest -> Dict Exercicio
    Resolucao -> Report
{ }
```

3.4 Modelo de Dados

Definiu-se que existirão três tipos de utilizadores: o administrador, o docente e o aluno/concorrente.

- Administrador - é a entidade com mais poder no sistema. É o único que pode criar contas

do tipo docente. É caracterizado por:

  - Nome de utilizador;
  - Nome completo;
  - Password
  - e-Mail

- Docente - entidade que tem permissões para criar concursos, exercícios, aceder aos resultados das submissões, (...). Os seus atributos coincidem com os do Administrador.
- Aluno/Concorrente - (...)

Decidiu-se que o sistema terá a noção de grupo. (...) O grupo é que possui as credenciais para entrar no sistema (nome de utilizador e password). Além disso também tem um nome pelo qual é identificado, um e-mail que será usado no caso de haver necessidade de se entrar em contacto com o grupo, e um conjunto de concorrentes.

Figure 3.2: Modelo de dados

Achamos importante incluir a informação de cada concorrente no grupo para, se possível, automatizar várias tarefas tais como lançamento de notas. Cada concorrente é caracterizado pelo seu nome completo, número de aluno (se for o caso), e e-mail.

Para finalizar vamos explicar em que consistem os concursos, exercícios e tentativas.
Um concurso, resumidamente, é um agregado de enunciados. Tem outras propriedades tais como um título, data de inicio e data de fim (período em que o concurso está disponível para que os grupos se registem), chave de acesso (necessária para o registo dos grupos), duração do concurso (tempo que o grupo tem para resolver os exercícios do concurso, a partir do momento que dá inicio à prova), e por fim, regras de classificação.

Um enunciado é um exercício que o concorrente tenta solucionar. Como seria de esperar, cada exercício pode ter uma cotação diferente, logo o peso do enunciado é guardado no mesmo. Para cada enunciado existe também um conjunto de inputs e outputs, que servirão para verificar se o programa submetido está correcto. Contém ainda uma descrição do problema que o concorrente deve resolver, assim como uma função de avaliação, função esta que define como se verifica se o output obtido está de acordo com o esperado.

Chapter 4

Conclusão e Trabalho Futuro

Contextualização...