Enunciado dos Requisitos

LAPR4
(Semana/Iteração 3)

(2012/13)

atb@isep.ipp.pt
v1.4 (2013-06-10)

Para todas as áreas funcionais / trilhos

Documentos técnicos necessários (que devem ser incluídos no relatório no javadoc)

- Casos de uso ou "user stories" (http://en.wikipedia.org/wiki/User_story);
- Diagramas de sequência ilustrando o "setup" da funcionalidade (i.e., extensão);
- Diagramas de sequência para ilustrar os *use case realization* de "análise"
- Diagramas de sequência para ilustrar de que forma os casos de uso/*user stories* vão ser realizados (design);
- Diagramas de classes ilustrando as novas classes e como elas "integram" com as classes existentes.
- Documentação sobre as classes.
- Documentação sobre testes unitários.

Nota Importante: As soluções devem seguir a arquitectura e padrões iniciais da aplicação. Em particular devem-se usar os ***pontos de extensão*** que já existem para acrescentar funcionalidades. Alterações estruturais devem ser devidamente justificadas no relatório técnico.

Notas sobre a avaliação:

- A nota de um aluno é maximizada pelo número diferente de áreas funcionais em que trabalhou. Assim, o ideal é que um aluno trabalhe em 3 áreas funcionais diferentes, uma por cada semana. Caso não seja possível deve tentar trabalhar em 3 trilhos diferentes.
- Deve ser clara a associação entre trilho/iteração e issue (um issue deve corresponder a apenas um trilho/iteração, mas pode haver mais do que um issue para um trilho/iteração).

Notas sobre os requisitos:

- Os requisitos são apresentados divididos em trilhos. Cada aluno deve escolher o trilho que será da sua responsabilidade nessa semana. Essa tarefa faz parte do planeamento (que devem fazer no início da semana com o apoio dos docentes de Gestão). Como resultado do planeamento espera-se que o grupo registe no Bitbucket os issues relativos ao planeamento semanal.
- Os requisitos podem ser, "naturalmente", incompletos e vagos. Para além disso os requisitos vão ser apresentados iterativamente, no início de cada semana. Os alunos devem usar os fóruns do moodle para tirarem dúvidas assim com podem consultar o regente (atb@isep.ipp.pt) nas suas aulas de LAPR4.

Área funcional: Persistência e I/O (EAPLI)
Trilho 1: *"Permitir importar e exportar dados, em particular de bases de dados, para facilitar a integração com aplicações terceiras."*

Iteração 1
Pretende-se que seja possível exportar o conteúdo de uma folha para uma tabela de uma base de dados.

De preferência deve existir uma nova opção de menu para efectuar esta funcionalidade. O utilizador deve indicar a área da folha que pretende exportar. Deve seleccionar a base de dados destino e, para essa base de dados, qual o nome da tabela que deve ser criada para receber os dados. A primeira linha de células a exportar indica o nome das colunas da nova tabela.

A aplicação deve suportar os mais variados drivers de JDBC de forma a ser possível exportar para diversos SGBDs relacionais. Em particular existe a necessidade imediata de suportar o HSQLDB (http://hsqldb.org/) que deve ser o SGBD sugerido por omissão.

Não deve ser usado nenhum mecanismo de ORM (Object-Relational Mapping) para este requisito.

Por uma questão de desempenho o processo de exportação dos dados deve ser executado paralelamente usando para isso uma nova *thread*.

Iteração 2
Pretende-se que seja possível importar o conteúdo de uma tabela de uma base de dados para uma folha. O processo é o "inverso" daquele proposto para a iteração1.

Por uma questão de desempenho o processo de importação de dados deve ser executado paralelamente usando para isso uma nova *thread*.

Na exportação deve ser possível acrescentar ou actualizar dados a uma tabela já existente (em lugar de criar uma tabela). Deve ser possível especificar qual uma combinação de colunas para "funcionar" como chave primária da tabela.

Deve ainda acrescentar-se o suporte para o SGBD Derby (http://db.apache.org/derby/).

Iteração 3
Pretende-se que um grupo de células possa ficar ligada de forma permanente a uma tabela de uma base de dados e periodicamente hajam actualizações (nos dois sentidos).
Para tal devem-se usar threads que executam regularmente (por exemplo a cada 30 segundos) a actualização entre as células e a tabela (sincronização).
O cleansheets, sempre que faz uma sincronização, guarda o estado das células após a sincronização numa estrutura de dados interna. Quanto executar a nova sincronização essa informação deve ser usada para determinar se:

- um registo foi removido nas células (deve ser removida na base de dados);
- um registo foi inserido nas células (deve ser inserido na base de dados);
- um registo foi apagado na base de dados (deve ser removido nas células);
- um registo foi inserido na base de dados (deve ser inserido nas células);
- um registo existe nos dois "lados" mas com valores diferentes (deve ser executado um "merge" das "colunas" alteradas. Caso a mesma coluna tenha sido alterada nos dos lados deve ser possível configurar qual a opção a tomar: prioridade cleansheets ou prioridade base de dados)

Trilho 2: *"Permitir novos formatos de persistência e armazenamento de dados como, por exemplo, o XML".*

Iteração 1
Definir um formato XML para persistência das folhas.

A persistência no novo formato XML deve ser tão completa quanto a persistência do formato original CLS. Por exemplo, pretende-se que este formato consiga armazenar também as propriedades que as extensões acrescentam às células.

Deve existir um *schema* (xsd) para descrever a estrutura do xml. Este deve ser usado para validar a abertura de folhas de cálculo no formato XML.

Devem ser usadas apenas as API de xml incluídas na plataforma Java 2 (javax.xml.*, org.w3c.dom.* e org.xml.sax.*). Não devem usar classes que gerem a representação dos objectos java em xml de forma automática (como por exemplo a classe XMLEnconder).

Iteração 2
Pretende-se implementar versões num novo formato de ficheiro (xml). A ideia é que quando se faz o "save" de um ficheiro seja guardada uma nova versão do worksheet "dentro" do mesmo ficheiro. Neste momento, quando se faz o "open" o cleansheets deve abrir a última versão que tiver no ficheiro. Ou seja, cada ficheiro xml vai conter várias versões "dentro" dele, uma para cada "save".

Para agilizar a implementação deste requisito foi decidido desistir da implementação xml da semana anterior. Em alternativa optou-se pela seguinte abordagem:
- Foi decidido (uma vez que o projecto já está a usar o HSQLDB para exportar e importar dados e que este SGBD permite criar base de dados totalmente em memória) que se deveria adoptar o HSQLDB para criar uma base de dados em memória que permitisse "conter" a representação de toda uma worksheet (ou seja, todos os dados que são persistidos com o "save"). Deve-se usar o hibernate para manipular esta base de dados em memória. Para persistir esta base de dados num ficheiro em formato xml deve-se usar o dbunit (www.dbunit.org).

Iteração 3
Pretende-se acrescentar uma nova extensão que permite que o utilizador possa mais facilmente gerir as versões associadas a um ficheiro. Essa extensão deve incluir uma "sidebar" que permita visualizar as versões existentes (mudando a versão "activa"). Se o utilizador fizer alterações numa versão "antiga" o cleansheets deve perguntar ao utilizador se deseja transformar essa versão na versão actual ou perder as alterações. Deve ainda permitir remover versões do ficheiro.

Em complemento a esta nova extensão, como requisito secundário era interessante que o cleansheets passasse a suportar o "undo" e "redo" de operações.

Área Funcional: Linguagens (LPROG)

Trilho 3: *"Incorporar novas funcionalidades na linguagem das fórmulas. Poderá ser necessário definir novas linguagens de fórmulas"*

Iteração 1

Pretende-se desenvolver uma linguagem nova de fórmulas.

As expressões (fórmulas) actuais que se podem executar numa célula começam pelo caractere '=' e os seus nomes são em inglês. Pretende-se implementar uma nova linguagem de fórmulas que inicie pelo caractere "#". A nova linguagem deve suportar as mesmas "funcionalidades" da linguagem actual.

Para além disso pretende-se para já a seguinte nova funcionalidade (para a nova linguagem):
- Pretende-se possibilitar a expressão de atribuição de valores a células. Por exemplo:
A2:=sum(A3:A10)
deve fazer o "sum" e atribuir o resultado à célula A2 sendo que o valor deve ser também o resultado da expressão (que é normalmente atribuído à célula actual). A sequência ':=' identifica uma atribuição.
- A nova linguagem deve ter um ficheiro de gramática próprio.

Deve-se seguir a abordagem actual do cleansheets. O código fonte da célula é "compilado" gerando uma árvore de parse (AST). A árvore de parse é depois percorrida resultando numa instância de uma classe que implementa a interface Expression. A interface Expression contempla o método "evaluate" que é executado para avaliar a "fórmula" e obter o seu resultado.

Nota:

- O cleansheets utiliza o ANTLR para gerar *parsers*. As gramáticas são descritas em ficheiros .g que dão origem a código java que implementa o parser.

Iteração 2

Pretende-se implementar sequências de expressões nas quais o resultado na célula é o da última expressão. Por exemplo:
#{ a1:=sum(A2:A11); a2:=if(a1>10; "grande"; "pequeno") }
Neste caso, se esta expressão for introduzida na célula b2, o que vai acontecer é que a célula b2 vai ficar com o mesmo valor que a célula a2.

A ideia é que as sequências de expressões funcionem como expressões n-árias, ou seja, expressões que têm um número indeterminado de parâmetros.

Para além de sequências simples como a que foi apresentada pretende-se suportar ciclos. Exemplo:
#whiledo{ a1>10; b2:=b2+a1; a1:=b1+1 }

Neste exemplo, a primeira expressão é a condição de "continuação". De seguida seguem-se uma ou mais expressões que devem ser executadas para cada iteração do ciclo. O resultado do whiledo é o resultado da última expressão executada (sem contar com a expressão de "continuação"). Em cada iteração, o whiledo testa primeiro a condição e só depois é que executa a iteração (se for caso disso).

Iteração 3

Pretende-se suportar uma nova variante de ciclos: o dowhile. Exemplo:
#dowhile{ b2:=b2+a1; a1:=b1+1; a1>10 }

Neste exemplo (dowhile), a última expressão da sequência é a condição de "continuação". Ao contrário do whiledo, neste caso, primeiro executam-se as expressões da iteração (i.e., todas excepto a última) e só depois a expressão da condição. O resultado do dowhile é o resultado da última expressão executada (sem contar com a expressão de "continuação").

Pretende-se a possibilidade de construir dinamicamente expressões. Por exemplo:
eval("#a2:=a3+1") – a função "eval" recebe uma string que interpreta como sendo o código fonte de uma fórmula. A função deve utilizar os compiladores de funções para compilar e depois executar a expressão. O resultado da expressão passa a ser o resultado da função "eval". A função eval deve permitir executar fórmulas das duas linguagens de fórmulas actuais: a "=" e a "#".

Trilho 4: *"Facilitar a customização do cleansheets pelos utilizadores finais. Uma linguagem de scripting pode ser necessária."*

Iteração 1
Deve-se iniciar o suporte a uma linguagem de macros.

A ideia é que as macros tenham um nome e sejam constituídas por uma sequência de instruções. Neste momento as instruções devem ser baseadas nas expressões que são possíveis de usar nas formulas das células.

Deve existir uma opção de menu que permite aceder a uma janela de macros. Essa janela deve permitir editar e executar macros. Por uma questão de desempenho o processo de execução de macro deve ser executado paralelamente (usando para isso uma nova *thread*).

Para já pretende-se que as macros suportem:

- as expressões que são possíveis de usar nas fórmulas das células;
- atribuir o resultado das expressões a células.
- Exemplo de macro:

```
A1:=sum(A1:A11)
A2:=if(A1>10; "grande"; "pequeno")
```

Deve-se reutilizar o código já existente para as fórmulas mas expandindo para ter a noção de "programa". Neste momento um programa é uma sequência de expressões (tipicamente uma por linha).

A nova funcionalidade de macros deve ser implementada numa nova extensão. Essa extensão deve acrescentar um "side bar" que inclua um mini editor de texto para o código da macro e um botão para executar a macro.

Iteração 2
Deve ser possível definir variáveis temporárias para conter resultados de instruções. Essas variáveis devem ter nomes começados pelo caractere "$".
Exemplo:
$resultado:=sum(A1:A11)
A2:=if($resultado>10; "grande"; "pequeno")

A janela de edição de macros deve apresentar o valor resultante da execução de uma macro numa nova área de "output". Considera-se que o resultado de execução de uma macro é o resultado da última instrução. Para além disso a janela deve permitir consultar o último valor que as variáveis temporárias assumiram na execução da macro.
A janela de edição de macros deve suportar editar várias macros. Cada macro é identificada por um nome. As macros devem manter-se em memória apenas enquanto a aplicação está "aberta".

Iteração 3

Deve ser possível invocar (i.e., chamar) uma macro. A sintaxe da instrução de invocação de macros pode ser: EXEC(<nome da macro>). O resultado da nova função Exec é o resultado da última instrução da macro que for executada.
Exemplo: exec(calcular_media)
Neste exemplo "calcular_media" é o nome de uma macro.

Outro objectivo será permitir que as macros possam ter parâmetros. Nesse caso estes devem poder ser acedidos localmente "dentro" da macro através da seguinte sintaxe: $1 (parâmetro 1), $2 (parâmetro 2), etc. A sintaxe de invocação das macros passa a ser EXEC(<nome da macro>, <parametro 1>, <parametro 2>, …).
Exemplo: exec(calcular_media, a1+b1, $primeiro, 100)
Neste exemplo, "calcular_media" é o nome da macro, "a1+b1" é uma expressão que resultará no valor do primeiro parâmetro. "$primeiro" é uma variável temporária que resulta no valor do segundo parâmetro e "100" é o valor do terceiro parâmetro.

Nota: As variáveis temporárias são locais às macros.

Área Funcional: Partilha de Folha (SCOMP e RCOMP)

Trilho 5: *"Permitir a edição partilhada de folhas de cálculo. A ideia é que instâncias diferentes do cleansheets possam partilhar parte do seu conteúdo"*

Iteração 1

Deve ser possível definir áreas de folhas de cálculo para partilha por outras instâncias do cleansheets.

A ideia é que uma instância do cleansheets funcione como origem da partilha e outra instância funcione como destino da partilha.

Na instância que funciona como origem da partilha o processo passa por definir uma porta para o serviço de partilha e qual a área da folha a partilhar.

Na instância que funciona como destino da partilha o processo implica introduzir o endereço e a porta da origem e indicar o canto superior esquerdo a partir do qual se deseja receber o conteúdo partilhado. Após conexão bem sucedida a instância destino recebe o conteúdo da área partilhada na origem e "apresenta" esse conteúdo a partir da célula que foi indicada.

A partilha de folha deve ser disponibilizada através do mecanismo de extensões do cleansheets. Deve existir uma "sidebar" (janela) com uma área que permita especificar os dados da partilha e outra área que permita especificar os dados da conexão a uma partilha.

A solução deve ser baseada numa arquitetura peer-to-peer de sockets em java.

Iteração 2

A partir do momento que uma partilha está activa em qualquer uma das instâncias do cleansheets pode-se mexer nessas células e as alterações são imediatamente visíveis nas outra instância conectada.

Para se fazer esta actualização automática nos dois sentidos deve-se usar o mecanismo de notificação das células e uma arquitectura baseada em threads.
Deve-se ter particular cuidado com questões de paralelismo nas *threads* e sincronismos no acesso aos objectos partilhados.

Deve implementar uma forma de encontrar na rede local instâncias de cleansheets que tenham partilhas activas. Esta funcionalidade evita que o utilizar necessite de saber o endereço das instâncias do cleansheets.

Iteração 3
Deve implementar um mecanismo simples de segurança para aceder a partilhas. Pode ser o conhecimento de uma palavra secreta que deve ser transmitida do cliente para o servidor. Deve tentar assegurar a confidencialidade da palavra secreta.

Cada instância do cleansheets pode ter várias partilhas. Cada partilha deve ter um nome.
O nome da partilha é dado por quem partilha. Vários clientes podem-se conectar a uma partilha.

Quando uma instância tenta ligar-se a uma partilha deve aparecer um diálogo com as partilhas existentes naquela instância para o utilizador seleccionar à qual se deseja ligar.

Deve ser possível activar e desactivar as partilhas. Quando uma partilha está desativada o servidor não deve aceitar clientes.
As partilhas devem também poder ser read-only ou writable. As read-only não aceitam actualizações dos clientes.

Na janela do "sidebar" devem aparecer a lista de partilhas que a instância disponibiliza (e o número de clientes conectados) e a lista de conexões que tem com outras instâncias.

Deve continuar a usar threads e os seus mecanismos de sincronização na sua solução.