universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Problema obrigatório 2

GAME OF THE ROPE

Jogo da corda

Curso	[8240] MI em Engenharia de Computadores e Telemática
Disciplina	[40814] Sistemas Distribuídos
Ano letivo	2015/2016

Alunos	[68021] Gabriel Vieira
	[68779] Rui Oliveira
Prática	P1
Grupo	05
Docente	Professor Óscar Pereira

Aveiro, 1 de Maio de 2016

Conteúdo

1 Introdução

Pretende-se através deste relatório apresentar os diagramas de interação do nosso sistema, sendo este um sistema distribuído por sete máquinas diferentes.

Para além do descrito acima, achámos conveniente apresentar uma pequena descrição da organização do nosso projeto, tipos de mensagem que são trocadas nesta implementação e uma breve descrição sobre cada método existente. Achámos também que seria relevante explicar o que fazem cada um dos *scripts* e *makefile* existentes.

2 Organização

De seguida iremos apresentar a organização em diretórios do nosso trabalho.

- **ClientSide**: é onde estão declaradas as entidades do sistema. É neste local que também estão todas as classes e interfaces que são necessárias para cada clientes. Cada entidade tem uma *main class* associada para que possam ser executadas em cada máquina.

- **Communication**: neste diretório encontram-se todas as classes que dizem respeito à comunicação, ou seja, ClientComm, ServerComm (disponibilizádas pelo professor Rui Borges em http://elearning.ua.pt/). Ainda dentro deste directório existe um sub-directório com as Mensagens (Message), tipos de mensagens (MessageType), proxy (ClientProxy e ServerInterface). Para além do descrito acima, existem dois módulos de constantes, são eles: ConstConfigs.java e CommConst.java.

- **ServerSide**: neste *package* estão inseridos os servidores e os interfaces necessários para a sua execução e interação com os respetivos clientes. Cada servidor tem uma *main class* associada para que possam ser executadas em cada máquina.

2.1 Makefile

O ficheiro *Makefile* permite compilar todo o código fonte (*.java) existente nos diretórios ClientSide, Communication e ServerSide. Os ficheiros *.class resultantes da compilação são adicionados na pasta __bin.

2.2 Script Bash (Unix shell)

2.2.1 script.sh

Permite executar a *Makefile* e posteriormente gerar os ficheiros JARs dos quatro servidores e dos três clientes (guardados na pasta JARS). Seguidamente, os ficheiros *.jar são enviados para as máquinas especificadas.

2.2.2 scriptReceiveLog.sh

Este script permite receber o ficheiro de *log* criado na máquina onde é executado o Repository. O ficheiro é guardado na pasta FilesLogs.

3 Descrição das Mensagens

Neste trabalho, usámos vários tipos de mensagens para cada tarefa, que vamos de seguida explicar a sua funcionalidade e os argumentos usados em cada:

Tipo Mensagem	**Argumentos**	**Possíveis Respostas**
CALL_ CONTESTANTS	ID do coach	ACK recebida com sucesso
CALL_TRIAL	Nº do jogo Nº do trial	ACK recebida com sucesso
IS_PLAYER_ SELECTED	ID do coach ID do jogador	POSITIVE caso seja selecionado NEGATIVE caso não seja selecionado
REVIEW_NOTES	ID do coach	ACK recebida com sucesso
SEAT_DOWN	ID do coach ID do jogador	ACK recebida com sucesso
ALL_SITING_ TEAMS		POSITIVE caso jogadores sentados NEGATIVE caso não sentados
FOLLOW_ COACH_ADVICE	ID do coach ID do jogador	ACK recebida com sucesso
END_OF_ THE_GAME	ID do coach	POSITIVE caso seja selecionado NEGATIVE caso não seja selecionado
TERMINATE		ACK recebida com sucesso

Tabela 1: Tipos de mensagem do Bench

Tipos mensagem	**Argumentos**	**Possíveis Respostas**
ANNOUNCE_ NEW_GAME	nº do jogo nº do trial	ACK recebida com sucesso
DECLARE_ GAME_WINNER	posição da corda	ACK recebida com sucesso
DECLARE_ MATCH_WINNER		ACK recebida com sucesso
END_OPER_ COACH	ID do coach	POSITIVE caso tenha terminado NEGATIVE caso não tenha terminado
TERMINATE		ACK recebida com sucesso

Tabela 2: Tipos de mensagem do Site

Tipos mensagem	**Argumentos**	**Possíveis Respostas**
START_TRIAL	nº do jogo nº do trial	ACK recebida com sucesso
ASSERT_TRIAL_ DECISION		DECISION_A Equipa A ganhou por knock out DECISION_B Equipa B ganhou por knock_out DECISION_C novo trial - jogo vai continuar DECISION_E jogo acabou, excedeu nº trials
INFORM_ REFEREE	ID do coach	ACK recebida com sucesso
GET_READY	ID do coach ID do jogador	ACK recebida com sucesso
AM_DONE	ID do coach ID do jogador força do jogador	ACK recebida com sucesso
SET_ POSITION_PULL	posição da corda	ACK recebida com sucesso
TERMINATE		ACK recebida com sucesso

Tabela 3: Tipos de mensagem do playground

Tipos mensagem	**Argumentos**	**Possíveis Respostas**
UPDATE_ REF_STATE	estado do referee	ACK recebida com sucesso
UPDATE_ COACH_STATE	ID do coach estado do coach	ACK recebida com sucesso
UPDATE_ CONTESTANT_STATE	ID do coach ID do jogador estado do jogador	ACK recebida com sucesso
UPDATE_STRENGTH	ID do coach ID do jogador força do jogador	ACK recebida com sucesso
UPDATE_AND_ WRITE_STRENGTH	ID do coach ID do jogador força do jogador	ACK recebida com sucesso
UPDATE_ PULL_POSITION	posição da corda	ACK recebida com sucesso
UPDATE_ TRIAL_NUMBER	nº do trial	ACK recebida com sucesso
UPDATE_ GAME_NUMBER	nº do jogo	ACK recebida com sucesso
IS_KNOCK_OUT	nº do jogo nº do trial nome das equipas	ACK recebida com sucesso
IS_END	nº do jogo nome das equipas	ACK recebida com sucesso
WAS_A_DRAW	nº do jogo	ACK recebida com sucesso
END_MATCH	nome das equipas resultado equipa A resultado equipa B	ACK recebida com sucesso
ADD_CONTESTANTS_ IN_PULL	ID do coach ID do jogador	ACK recebida com sucesso
REMOVE_CONTESTANTS _IN_PULL	ID do coach ID do jogador	ACK recebida com sucesso
TERMINATE		ACK recebida com sucesso

Tabela 4: Tipos de mensagem do repository

4 Repositório de desenvolvimento

https://github.com/ruipoliveira/gameoftherope.git

5 Tipo de dados

5.1 Entidade ativa: instanciado e com um fio de execução

5.1.1 Árbitro (Referee)

- **Estrutura de dados interna**

  state: variável onde é guardado o estado actual do arbitro.

- **Operações**

  setState(ERefereeState state) - permite atualizar o estado do arbitro

  RefereeState getCurrentState() - permite aceder ao estado do arbitro

5.1.2 Treinador (Coach)

- **Estrutura de dados interna**

  state: variável onde é guardado o estado actual do treinador.

- **Operações**

  setState(ECoachesState state) - permite atualizar o estado do treinador

  ECoachesState getCurrentState() - permite aceder ao estado do treinador

  int getIdCoach() - Permite aceder ao identificador do treinador.

5.1.3 Jogador (Contestant)

- **Estrutura de dados interna**

  state: variável onde é guardado o estado actual do jogador.

  contestStrength: variável onde é guardado a força de cada jogador. Esta força é obtida através da função generateStrength()

- **Operações**

  setState(EContestantsState state) - permite atualizar o estado do jogador

EContestantsState getCurrentState() - permite aceder ao estado do jogador

int getIdCoach() - Permite aceder ao identificador do treinador.

int getStrength() - Permite aceder à força do respetivo jogador.

int generateStrength() - Permite gerar de forma aleatória a força associada a cada jogador. Valor da força é um inteiro entre 10 e 20.

5.2 Entidade passiva: instanciado e sem um fio de execução

5.2.1 Banco (Bench)

- **Operações**

callContestants(int coachId) - permite que os treinadores chamem os jogadores para campo. Nesta mesma função o treinador irá esperar que os jogadores estejam pronto no banco, de seguida irá escolher três deles para ir jogar (o método de escolha é aleatório). Seguidamente, o treinador irá esperar que os jogadores estejam prontos na corda.

callTrial(int numGame, int numTrial) - para o primeiro trial o arbitro irá informar os treinadores para se prepararem para um novo trial. No caso dos restantes trials, o jogador irá esperar que os treinadores terminem a operação reviewNotes().

isPlayerSelected(int coachId, int contestId) - verifica se um jogador foi escolhido para se deslocar para a corda.

reviewNotes(int coachId) - O treinador aguarda que os jogadores se sentem, caso isso aconteça a estrutura de dados que armazena os jogadores na corda é limpa.

seatDown(int coachId, int contestId) - os jogadores esperam que os restantes jogadores já tenham acabado a simulação de puxar a corda, caso isso aconteça os jogadores irão sentar-se.

allSittingTeams() - verifica se todos os jogadores estão sentados.

followCoachAdvice(int coachId, int contestId) - os jogadores aguardam que sejam chamados pelo seu treinador para irem jogar um novo trial. No fim do mesmo acontecer, os jogadores irão informar o respetivo treinador que estão prontos.

endOfTheGame() - permite verificar quando é que um jogo acaba ou não.

5.2.2 Local do árbitro (Site)

- **Operações**

  announceNewGame(int numGame, int nrTrial) - funcao de transição de estado. Apenas é usada para guarda o numero do jogo e do trial neste monitor.

  declareGameWinner(int posPull) - permite definir qual das equipas é a vencedora do respetivo jogo. O resultado é obtido através da observação da posição da corda.

  declareMatchWinner - define qual das equipas é a vencedora do match. Verifica qual das equipas ganhou o maior numero de jogos.

  endOperCoach(int id) - permite verificar quando é que um dado treinador termina o seu processo.

5.2.3 Campo (Playground)

- **Operações**

  startTrial(int nrGame,int nrTrial) - permite dar inicio a um novo trial. Inicialmente o arbitro irá esperar que os treinador estejam pronto e posteriormente irá informar os jogadores que podem iniciar a partida.

  assertTrialDecision() - está função é executa pelo arbitro no final de cada trial. Inicialmente o arbitro aguarda que todos os jogadores tenham puxado a corda. Posteriormente, irá somar a força de todos os jogadores de cada equipa e acumulá-lo numa variável. Seguidamente irá decidir quem vence o trial, se fica empatado ou se existe knockOut.

  informReferee(int coachId) - treinador informa arbitro que estão prontos.

  getPositionPull() - retorna posição da corda

  setPositionPull(int posPull) - atualiza posição da corda

  getReady(int coachId, int contId) - jogadores esperam que o arbitro inicie o trial .

  amDone(int coachId, int contId, int contestStrength) - nesta função é simulado o puxo da corda.

5.2.4 Repositório geral

- **Operações**

  initWriting() - escreve cabeçalho num ficheiro.

writeLine() - escreve cada linha num ficheiro

endWriting() - termina processo de inscrita no ficheiro.

writeLineGame() - escreve primeira linha no inicio de cada jogo e repete cabeçalho.

Funções de atualização de estados: -*updateRefState(ERefereeState state)* -*updateCoachState(int idCoach, ECoachesState state)* -*updateRefState(ERefereeState state)*

updateStrength(int idTeam, int idContest, int contestStrength) - função de atualização da força de um determinado jogador. É usada no construtor de um jogador.

updateStrengthAndWrite(int idTeam,int contestId, int contestStrength) - similar à anterior mas permite escrever uma nova linha num ficheiro.

updatePullPosition(int posPull) - atualiza a posição atual da corda

updateTrialNumber(int nrTrial) - atualiza o número atual do trial

updateGameNumber(int nrGame) - atualiza o número atual do jogo

isKnockOut(int nrGame, int nrTrial, String team) - se existir knockOut esta função é invocada. Permitirá escrever uma linha indicando o mesmo.

isEnd(int nrGame, String team) - se o jogo terminar esta função é invocada. Permitirá escrever uma linha indicando o mesmo.

wasADraw(int nrGame) - se o jogo ficar empatado esta função é invocada. Permitirá escrever uma linha indicando o mesmo.

endMatch(String team, int resultA, int resultB) - se o match terminar esta função é invocada. Permitirá escrever uma linha indicando o mesmo.

6 Diagrama de interação

Foi considerado o esquema introduzido durante as aulas teóricas para a representação do diagrama de interação.

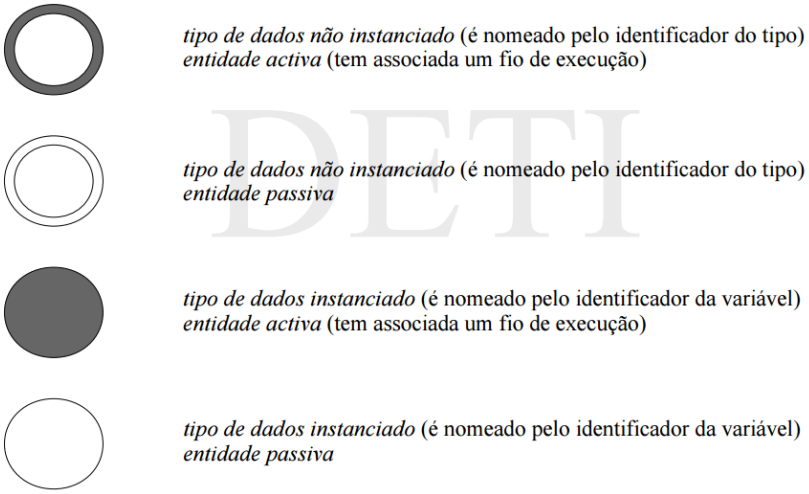

Figura 1: Elementos base dos diagramas de interação

6.1 Servidores

6.1.1 Bench

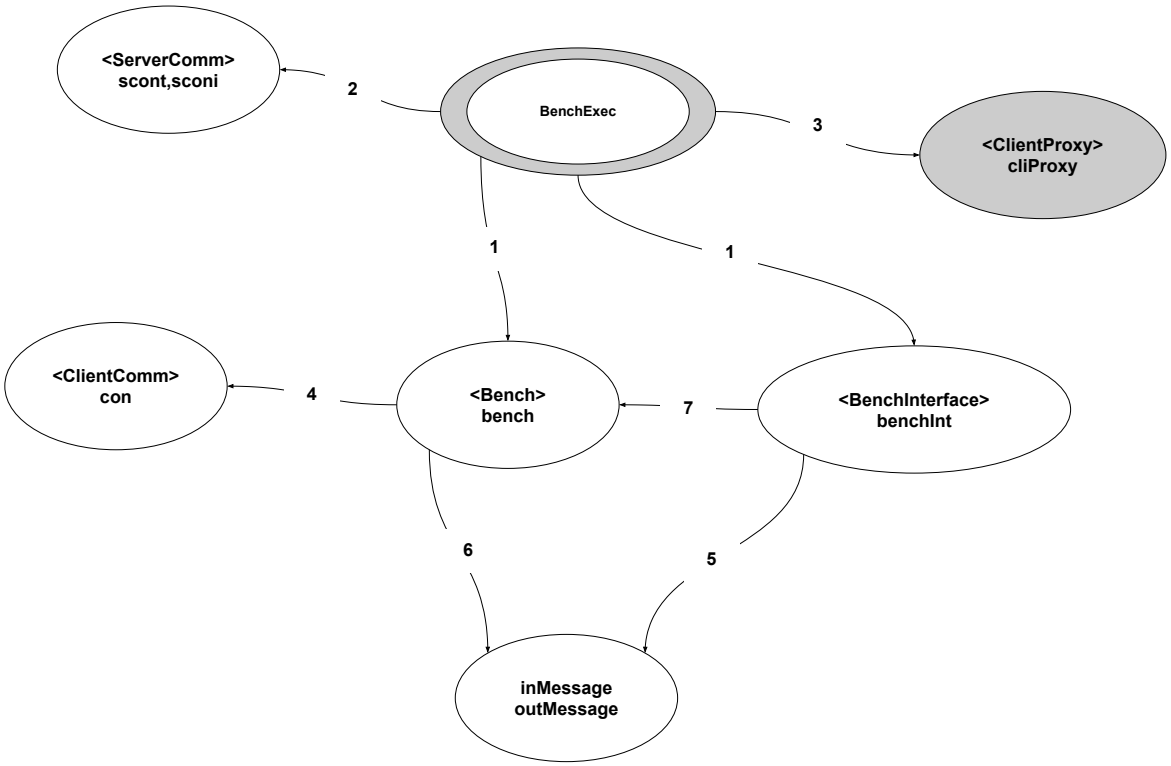

Figura 2: Diagrama de interação

1. instantiate
2. instantiate, start, accept
3. instantiate, start
4. instantiate, readObject, writeObject
5. instantiate, getType, getIdCoach, getGameNumber, getTrialNumber, getIdContest
6. instantiate, getType
7. callContestants, callTrial, isPlayerSelected, reviewNotes, seatDown, allSittingTeams, followCoachAdvice, endOfTheGame

6.1.2 Playground

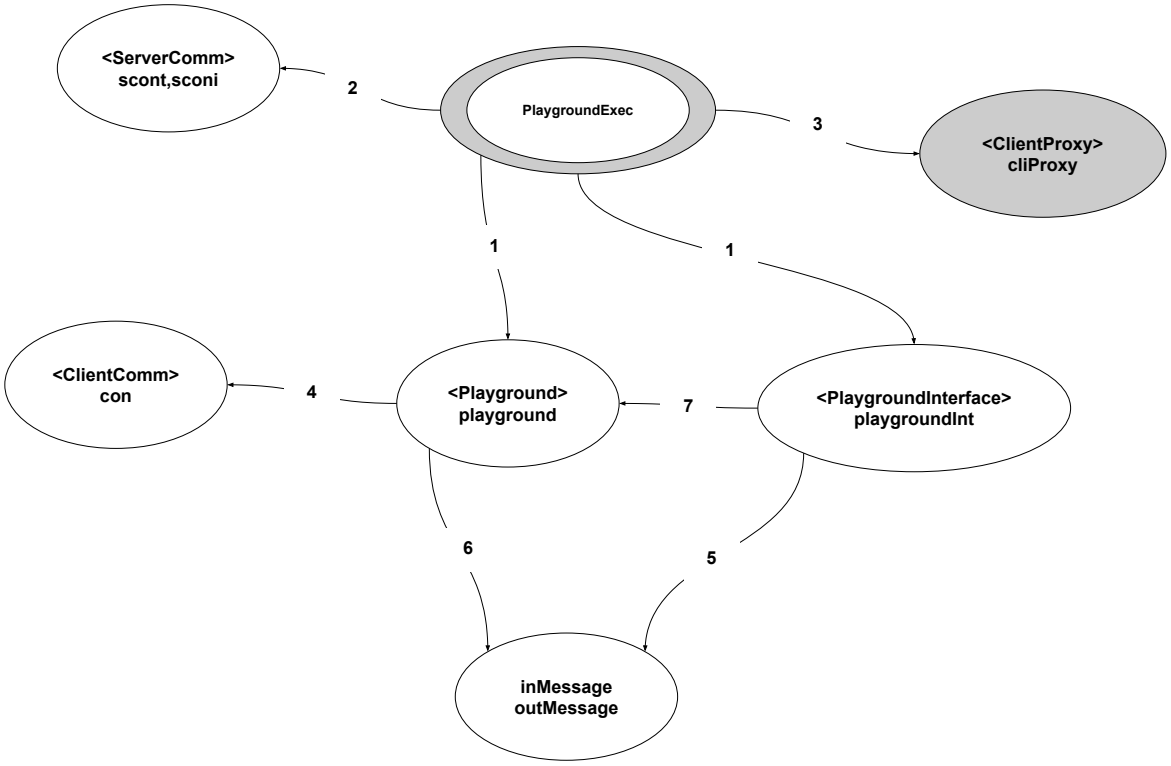

Figura 3: Diagrama de interação

1. instantiate
2. instantiate, start, accept
3. instantiate, start
4. instantiate, readObject, writeObject
5. instantiate, getType, getGameNumber, getTrialNumber, getIdCoach, getIdContest, getContestStrength, getPositionPull, posPull
6. instantiate, getType
7. startTrial, assertTrialDecision, informReferee, setPositionPull, getReady, amDone

6.1.3 Site

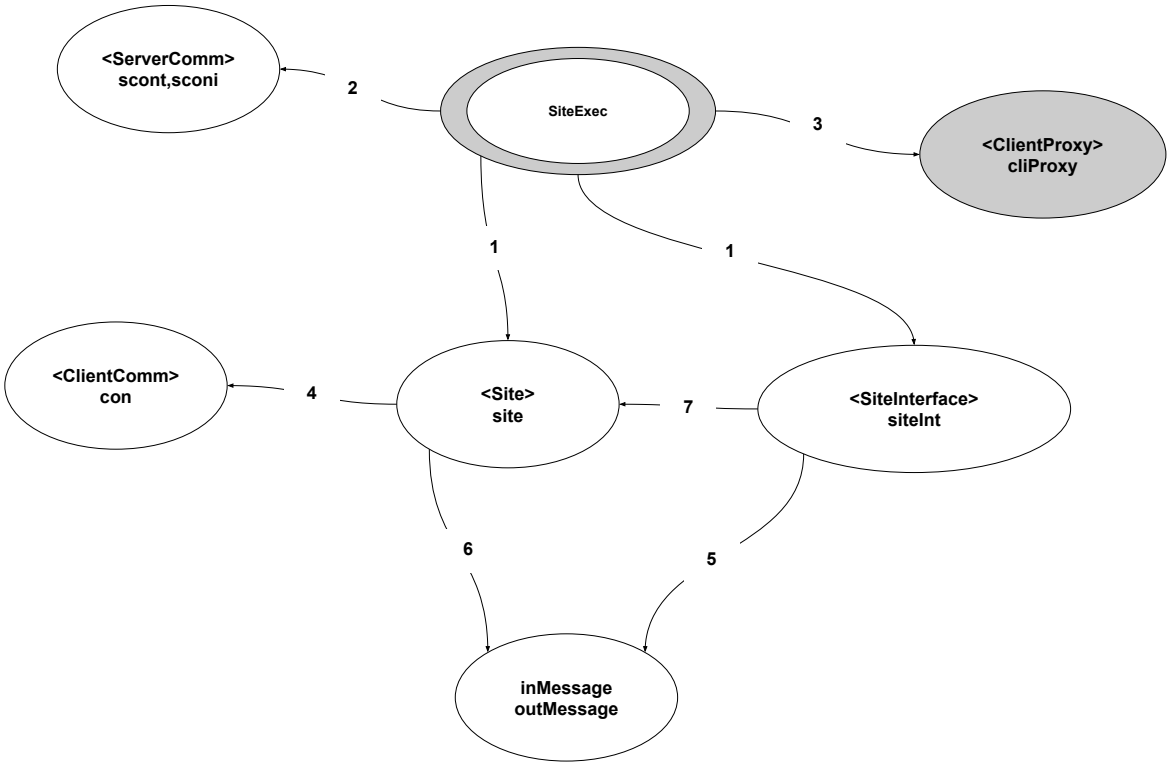

Figura 4: Diagrama de interação

1. instantiate
2. instantiate, start, accept
3. instantiate, start
4. instantiate, readObject, writeObject
5. instantiate, getType, getGameNumber, getTrialNumber, getPullPosition, getIdCoach
6. instantiate, getType
7. announceNewGame, declareGameWinner, declareMatchWinner, endOperCoach

6.1.4 Repository

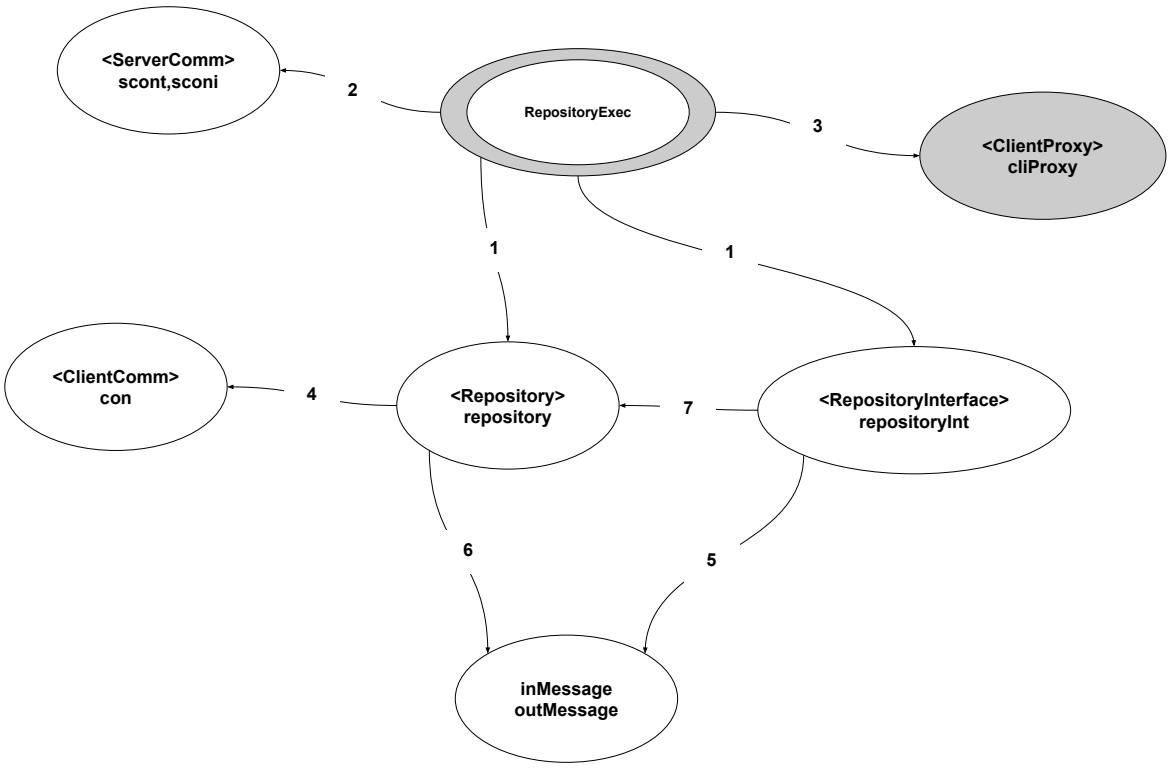

Figura 5: Diagrama de interação

1. instantiate
2. instantiate, start, accept
3. instantiate, start
4. instantiate, readObject, writeObject
5. instantiate, getType, getRefState, getIdCoach, getCoachState, getIdContest, getContestState, getContestStrength, getPullPosition, getTrialNumber, getGameNumber, getTeam, getResultA, getResultB
6. instantiate, getType

7. updateRefState, updateCoachState, updateContestantState, updateStrength, updateStrengthAndWrite, updatePullPosition, updateTrialNumber, updateGameNumber, isKnockOut, isEnd, wasADraw, endMatch, addContestantsInPull, removeContestantsInPull

6.2 Clientes

6.2.1 Referee

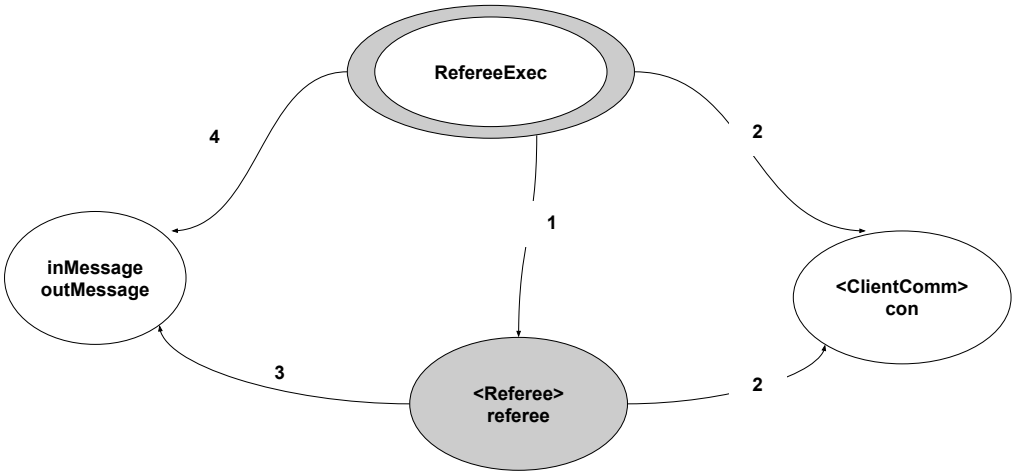

Figura 6: Diagrama de interação

1. instantiate, start, join
2. instantiate, writeObject, readObject
3. instantiate, getType, getRefState (CurrentState), getPullPosition
4. instantiate, getType

6.2.2 Coach

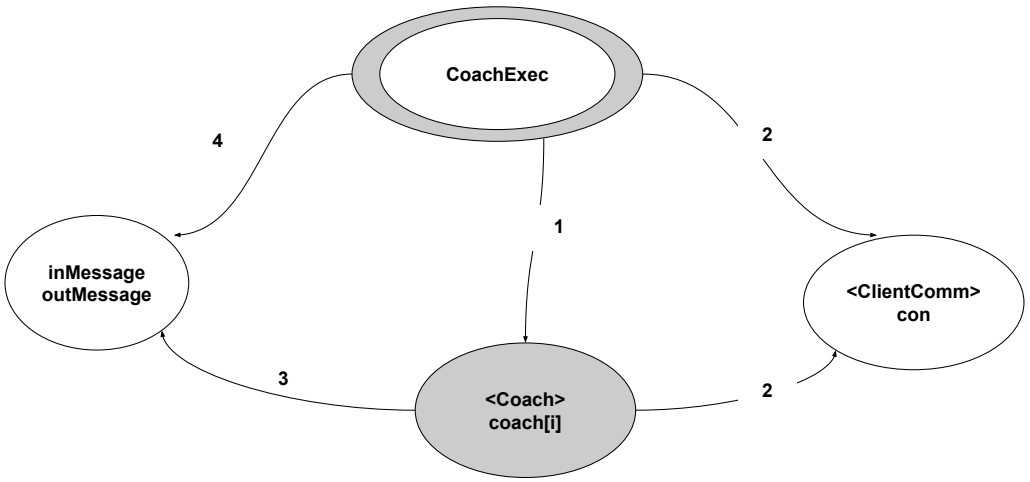

Figura 7: Diagrama de interação

1. instantiate, start, join
2. instantiate, writeObject, readObject
3. instantiate, getType
4. instantiate, getType

6.2.3 Contestant

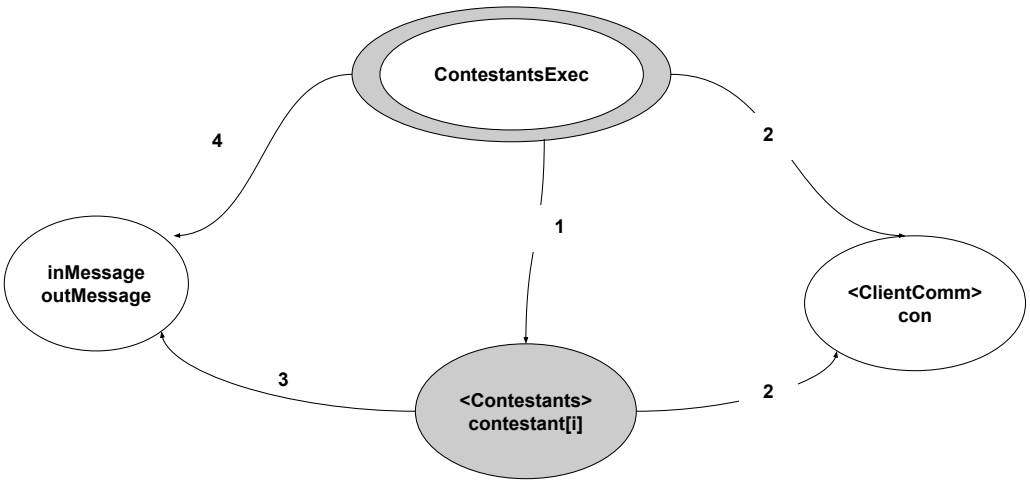

Figura 8: Diagrama de interação

1. instantiate, start, join
2. instantiate, writeObject, readObject
3. instantiate, getType
4. instantiate, getType