Table of Contents

1. Compiling and running Server & Avatar: 2

1.1 Server: Building and running the server. 2

1.2 Avatar: 2

2. Managing Tournaments & Users 3

2.1 Tournament Management: 3

2.2 Configuration files: 3

2.3 User Management: 5

3. Making a clever avatar: 5

3.1 Understanding the terminologies: 6

3.1.1 Refutation 6

3.1.2 Strengthening 6

3.1.3 Agreement 7

3.2 CodeSamples 8

3.1.1. Avatar's Class Dictionary template: 8

3.1.2. Avatar's Behavior definition template: 9

4. Smart History: 11

4.1. Understanding Smart History files: 11

1. Compiling and running Server & Avatar:

1.1 Server: Building and running the server.

Step 1: Execute build.xml:

Location: /GenericSCG

Command: ant

Step 2: Run the server

Location: GenericSCG/bin

Command: java -cp .:demeterf.jar:hamcrest-all-1.3.0RC2.jar scg.admin.Admin <admin password>

1.2 Avatar:

Step 1: Generate Java files using Demeterf

Location: GenericSCG

Command: java -cp .:demeterf.jar:hamcrest-all-1.3.0RC2.jar demeterf <./src/mmg/avatar/mmgAvatar.cd> <./src/mmg/avatar/mmgAvatar.beh> <gen>

The command shown above is for mmg avatar.

Step 2: Build the source files

Location: /GenericSCG

Command: ant

Step 3: Run the avatar

Location: /GenericSCG/bin

Command: java -cp .:demeterf.jar:hamcrest-all-1.3.0RC2.jar scg.net.avatar.PlayerMainMMG <random-port> <server-name> <team-username> <team-password> <tournamentID>

The command shown above is for mmg avatar.

2. Managing Tournaments & Users

These are the following steps to manage users and to start a new tournament.

2.1 Tournament Management:

1. Once the server is up and running, open the URL http://server-url:7007/signin (example: http://localhost:7007/signup, incase if server is running locally)
2. Enter the username: root and password: password given while executing Admin class
3. Create a new tournament by filling in all the required fields.
    a. Please refer section 2.2 to get the config file for a particular playground.
4. All the users who are willing to participate in the tournament must enroll to a particular tournament and then run their avatar (Step 3 of 1.2 should be done after enrolling into a tournament)

2.2 Configuration files:

The below configuration has to be used while creating the tournaments. Configuration is specific to a playground.

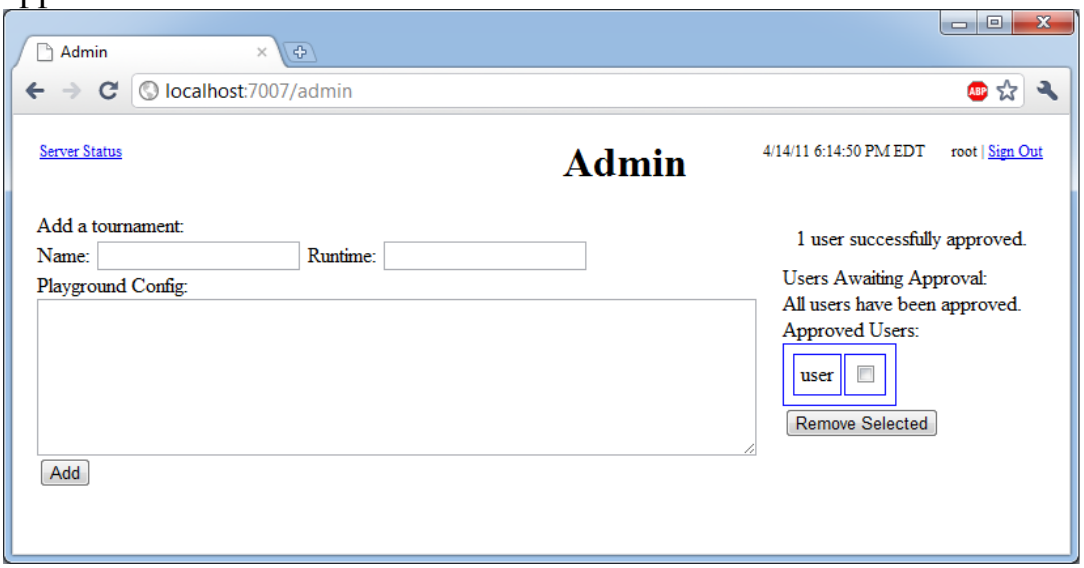
1. MMG:
   scg_config[
   domain:mmg.MMGDomain
   protocols: scg.protocol.ForAllExistsMax
   tournamentStyle: full round-robin
   turnDuration: 60 //seconds
   maxNumAvatars: 20
   minStrengthening: 0.001
   initialReputation: 100.0
   maxReputation: 1000.0
   reputationFactor: 0.4
   minProposals: 2
   maxProposals: 5
   numRounds: 6
   proposedClaimMustBeNew: true
   minConfidence: 0.5
   ]
   mmg.MMGConfig {{ mmg_config[ ] }}

2. BSF:
   config[
   domain:bfs.BFSDomain
   protocols: scg.protocol.ForAllExistsEqual
   tournamentStyle: full round-robin
   turnDuration: 60 //seconds
   maxNumAvatars: 20
   minStrengthening: 0.001
   initialReputation: 100.0
   maxReputation: 1000.0
   reputationFactor: 0.4
   minProposals: 2
   maxProposals: 5
   numRounds: 6
   proposedClaimMustBeNew: true
   minConfidence: 0.5
   ]
   bfs.BFSConfig {{ bfs_config[ ] }}

3. HSR:
   HSR config
   scg_config[
   domain:hsr.HSRDomain
   protocols: scg.protocol.ForAllExistsMax
   tournamentStyle: full round-robin
   turnDuration: 60 //seconds
   maxNumAvatars: 20
   minStrengthening: 0.001
   initialReputation: 100.0
   maxReputation: 1000.0
   reputationFactor: 0.4
   minProposals: 2
   maxProposals: 5
   numRounds: 6
   proposedClaimMustBeNew: true
   minConfidence: 0.5
   ]
   hsr.HSRConfig {{ hsr_config[ ] }}

2.3 User Management:

1. Sign up: Open the URL http://server-url:7007/signup to sign-up and wait until the Admin approves your request.
2. Sign In: Once the admin has approved the sign up request, user can login. The URL for login page is http://server-url:7007/signin
3. Approve/Remove users:
   Administrator can approve or remove users directly from the admin control panel. After logging in, pending users (i.e., "users awaiting approval") will be shown on the right . Additionally, the administrator can elect to remove users that had previously been approved.



3. Making a clever avatar:

The students would be provided with the baby avatar i.e., a .cd and .beh file. For example, these two files for MMG playground is located under mmg.avatar package in GenericSCG .

Step 1: No changes required for .cd file.

Step 2: .beh file has following methods, which need to be modified to make a clever avatar:

List<Claim> **propose**(List<Claim> forbiddenClaims): The "propose" method is used to make new claims during competitions.

List<OpposeAction> **oppose**(List<Claim> claimsToBeOpposed): The "oppose" method is used to respond to the claims of the proposer. The claims are given in the input parameter claimsToBeOpposed.

InstanceI **provide**(Claim claimToBeProvided):
The "provide" method is used to provide an instance for the given claim.

SolutionI **solve**(SolveRequest solveRequest):
The "solve" method is used to provide solution for the instance provided by opposition.

3.1 Understanding the terminologies:

3.1.1 Refutation

Alice makes a claim C using protocol P with quality qA and confidence cfA. Bob refutes C. The protocol P specifies the sequence of actions that are to be to be performed by Alice and Bob.

Depending on the protocol, either Alice or Bob (or both) will provide an instance to be solved and a solution for the given instances of claim C. The quality of the solution(s) and quality qA will be used by the protocol predicate (i.e., getResult) to determine the outcome of the refutation. The result of this predicate is a value between 1 and -1. This result is used in computing the updates to the reputation of the two avatars.

If Bob successfully refutes the claim, Bob wins reputation and Alice loses reputation. If Alice successfully defends her claim, Alice wins reputation and Bob loses reputation.

The reputation is updated as below:
Alice's new reputation = Alice's current reputation + (cfA * result)
Bob's new reputation = Bob's current reputation - (cfA * result)

3.1.2 Strengthening

Alice makes a claim C using protocol P with quality qA and confidence cfA.
Bob strengthens claim C with quality qB and cfB where cfB >= cfA.
Alice refutes this strengthened claim. The refutation follows the steps specified in the protocol P.

If Bob successfully defends his strengthened claim, Bob wins and the reputation updates are as follows:
Bob's new reputation = Bob's current reputation + (cfA * | qA – qB |)
Alice's new reputation = Alice' current reputation - (cfA * | qA - qB |)

If Bob fails to defend his strengthened claim, Alice wins and the reputation updates are as follows:
Alice's new reputation = Alice's current reputation + cfB
Bob's new reputation = Bob's current reputation – cfB

3.1.3 Agreement

Alice makes a claim C using protocol P with quality qA, confidence cfA. When Bob agrees on claim C with Alice, the following conditions should hold true:
Bob must defend C against Alice.
Bob must refute !C (i.e., the negated claim of C). !C has the same InstanceSet, quality and confidence as C but has a protocol !P with Alice as defender.

If Bob fails to satisfy any one of the above condition, then Bob loses.
Similarly Alice must satisfy the following conditions:
Alice must defend C against Bob.
Alice must refute !C with Bob as the defender.
If Alice fails to satisfy any one of the above condition, then Alice loses.

If Bob loses, the reputation is updated as follows:
Bob's new reputation = Bob's current reputation – cfA
Alice's new reputation = Alice's current reputation + cfA
If Alice loses, the reputation is updated as follows:
Alice's new reputation = Alice's current reputation – cfA
Bob's new reputation = Bob's current reputation + cfA
If both Alice and Bob satisfy all their conditions, the reputations remain unaffected and the claim goes into the social welfare set (i.e., the claim repository).

3.2 CodeSamples

The code samples below represent a template class dictionary (i.e., “ddsAvatar.cd”) and a template behavior definition (i.e., “ddsAvatar.beh”) needed for a complete avatar definition.

During actual implementation, all the instances of “dds” should be replaced with an acronym of the playground. For example, HSR playground class should be named as “hsrAvatar”. Note that file names should also reflect this change. Therefore, the resulting files in this example would be named “hsrAvatar.cd” (for the class dictionary) and “hsrAvatar.beh”.

3.1.1. Avatar’s Class Dictionary template:

```
/*  File: ddsAvatar.cd
        (replace all instances of dds, including the file name, with a three
        letter representation of the domain name) “scg.cd” includes the class
        definitions for SCG courts. The SCG class dictionary has meta-level
        definitions that specify how each avatar definition cd must be defined
        */

nogen include "../scg/scg.cd";

   /* This avatar definition will be created within the * dds.Avatar package.*/

package dds.Avatar;

/** - Import the SCG package.* (necessary to ensure that this avatar is
compatible with the * SCG courts system)* - Import the DDS package.*
(necessary to ensure that this avatar can interact with* dds specific
objects)After importing the SCG and DDS packages, you should also import
any additional packages or classes that will be required for the
implementation of methods in your .beh file */

import scg.*;

import dds.*;

// Enter additional “import” statements before this comment.

/*  Define a domain design specific Avatar (i.e., ddsAvatar). The ddsAvatar
definition must implement the AvatarI interface. Therefore, AvatarI methods
```

must be defined in your corresponding behavior file (.beh) as part of the ddsAvatar class (see the template avatar behavior definition for more details). */

ddsAvatar = implements AvatarI.

3.1.2. Avatar's Behavior definition template:

/*File: ddsAvatar.beh
(replace all instances of dds, including the file name, with a three letter representation of the domain name) Methods for the ddsAvatar class.

The ddsAvatar class must implement the AvatarI interface. Therefore, the „propose", „oppose", „provide", and „solve" * methods must be implemented.*/

ddsAvatar {{
/*The constructor for the ddsAvatar class contains a Config object.*/

private Config config;
/*The constructor to be called during registration (where you * supply a Config)*/

public ddsAvatar(Config cfg)
{ config = cfg; }
/*Proposes a List<Claim> that does not include any claims from the given List<Claim> (i.e., forbidden claims) */

public List<Claim> propose(List<Claim> forbiddenClaims) { // Replace with domain specific logic:

return null;

}

// Must return a "List<Claim>" object

// Decides what opposition action to take for each claim in the given

```
List<Claim> public List<OpposeAction> oppose(List<Claim>
claimsToBeOpposed){

// Replace with domain specific logic:

return null;

// Must return a “List<OpposeAction>” object

}

// Provides a ddsInstance for the given Claim public InstanceI provide(Claim
claimToBeProvided) {

// Replace with domain specific logic:

return null;

// Must return a “ddsInstance” object

}

// Solves (i.e., gives a ddsSolution) for the instance in the given
SolveRequest

public SolutionI solve(SolveRequest solveRequest)

{

ddsInstance i = (ddsInstance)solveRequest.getInstance();

// Replace with domain specific logic:

return null;

// Must return a „ddsSolution“ object

    }
    /*Include helper methods for the “propose”, “oppose”, “provide”, and/or
        “solve” methods here (i.e., before the double curly* brackets)*/
    }}
```

4. Smart History:

4.1. Understanding Smart History files:

Consider a sample paragraph of the smart history file from a MMG game. Let us try and understand each line and field means.

SAMPLE 1:

claim mmg.MMGInstanceSet {{ }} scg.protocol.ForAllExistsMax {{ }} 0.5707252354898215 1.0
proposer {{ navi }}
opposer {{ dexter }}
action strengthening 0.5807252354898215
responses provider {{ navi }} pr provide mmg.MMGInstance {{ 0.05 }}
provider {{ dexter }} pr solve mmg.MMGSolution {{ 0.046511853922261426 }}
winner {{ dexter }}
pointsWon 1.0

SAMPLE 2:

claim mmg.MMGInstanceSet {{ }} scg.protocol.ForAllExistsMax {{ }} 0.106 1.0
proposer {{ dexter }}
opposer {{ navi }}
action agree
responses provider {{ navi }} pr provide mmg.MMGInstance {{ 0.05 }}
provider {{ dexter }} pr solve mmg.MMGSolution {{ 0.4648488775874373 }}
winner {{ dexter }}
pointsWon 1.0

SAMPLE 3:

claim mmg.MMGInstanceSet {{ }} scg.protocol.ForAllExistsMax {{ }} 0.7323630210011601 1.0
proposer {{ navi }}
opposer {{ dexter }}
action refuting
responses provider {{ navi }} pr provide mmg.MMGInstance {{ 0.3 }}
provider {{ dexter }} pr solve mmg.MMGSolution {{ 0.3158511574800351 }}
winner {{ navi }}
pointsWon 1.0

KEY:

claim INSTANCE SET PROTOCOL QUALITY CONFIDENCE
proposer {{ AVATAR_NAME }}
opposer {{ AVATAR_NAME }}
action ACTION NAME: REFUTE/STRENGTHEN/AGREE STRENGTHENED CLAIM(if action is strengthening)
responses provider {{ AVATAR_NAME }} pr FUNCTION CALLED INSTANCE {{ INSTANCE VALUE }} provider {{ AVATAR_NAME }} pr FUNCTION CALLED SOLUTION {{ SOLUTION VALUE }}
winner {{ AVATAR_NAME }}
pointsWon VALUE

EXPLANATION:

Consider sample 1. It represents the history of first round out of the 9 rounds (MAxrounds) between team navi and team dexter
- team navi proposes with a claim of C = 0.5707252354898215
- team dexter opposes by strengthening 0.5807252354898215
- team navi provides with a value of x= 0.5
- team dexter solves with a value of y= 0.046511853922261426
- team dexter wins this round winning 1.0 points.