"Generic SCG Courts" - System Design Document

CS 5500 Project 2

Team-Members: Parvathy Unnikrishnan Nair, Reto Kleeb, Xinyi Wang

Avatar: PRX

1. Introduction

The SCG Courts system has been developed for participants to play SCG games against each other. This system can be used for measuring the quality of an avatar as well as finding better solutions to the games being played.

1.1. Purpose of the System

The purpose of this new implementation of the SCG Courts system is, to allow participants to compete against each other in Scientific Community Games. The system allows multiple games, following individual rules to be played in tournaments. The results of these tournaments can be used in multiple ways: First of all, the results of a tournament offer the participants and observers an objective measurement of the quality of an avatar in comparison to the other participants in a tournament. The second information that can be gathered from the results of a tournament are, the actual solutions to the problems that were offered, challenged and possibly re-challenged during the tournament. The existing system consists of a server and an example player with minimal intelligence to play the game according to the protocol. The game, together with some examples is introduced on [SCGPG].

1.2. Terminology

Each game that is played in an SCG court is defined via a specific set of elements that are then used in the context of a generic court:

Domain	The domain is the set that consists all of the required parts to represent a complete, functional SCG court game (Instance, Solution, valid, quality). Examples are HSR, MAX-CSP
Instance	A instance represents a set of concrete instances of a problem in the given domain. For the HSR game a possible instance could be HSR(9, 2) > 5
Solution	A solution contains the data that has been generated by a participant that claims to have a solution or by a participant that was forced to come up with a solution to prove that a earlier made claim was valid. Solutions have to be verifiable (see 'Valid"). An example in the HSR game would be a binary tree of a certain height.
Valid	The valid function takes an instance and a solution and returns a Boolean answer to the question: Was the given solution a valid answer for the given instance? In the HSR game the valid function would verify if a binary tree 'bt' had a height of 5 or less if the claim was made with the

	instance HSR(9, 2) <= 5
Quality	The quality function stongly depends on the game that is played. The function describes how many percent of a solution that is considered perfect, have been achieved by the given solution. The function returns a value between 0 and 1. In the MAX-CSP example the quality is defined as the number of satisfied (weighted) constraints divided by the total of (weighted) constraints
Strengthen	The strengthen(C1, C2) function returns true if C2 is stronger than C1

1.3. Design goals

The SCG Courts system has been designed to satisfy the following set of non-functional requirements:

- The system was implemented for easy operation at low cost
- The maximum response time of the system is minimized so that the player responds to the server in a particular time frame
- The system also has been designed with modifiability in mind, since a better algorithm or a new game should be easy to incorporate
- Reliability - A single client should not affect tournament. The tournament continues even when a client leaves
- Dependability - The system kicks out clients which do not follow the guidelines
- Security of the system is guaranteed as the server keeps track of the scores, scores are modified by actual challenges and cannot be manipulated

The trade-offs that were faced during the implementation are the following:

Trade-off	**Rationale**
Efficiency v. Portability	The source code was written in Java and hence was not as efficient as a code written in languages like C, but this was traded off with a better portability.
Backward Compatibility v. Readability	The current implementation is based on a DemeterF generated protocol, for a new version the desire to use XML or JSON based protocols might come up. Such an approach would simplify the communication with non Java/C# based Avatars. Introducing these breaking changes however would break the backward compatibility of the system.

2. Current software architecture

The current implementation of the SCP Court is based on a classical client / server architecture. This architecture allows to distribute the load in CPU intense competitions and therefore speeds up the tournament time. The central server acts as a trusted third party that verifies that all participants follow the protocol and takes care of the scoring.

2.1. Issues with the current Implementation

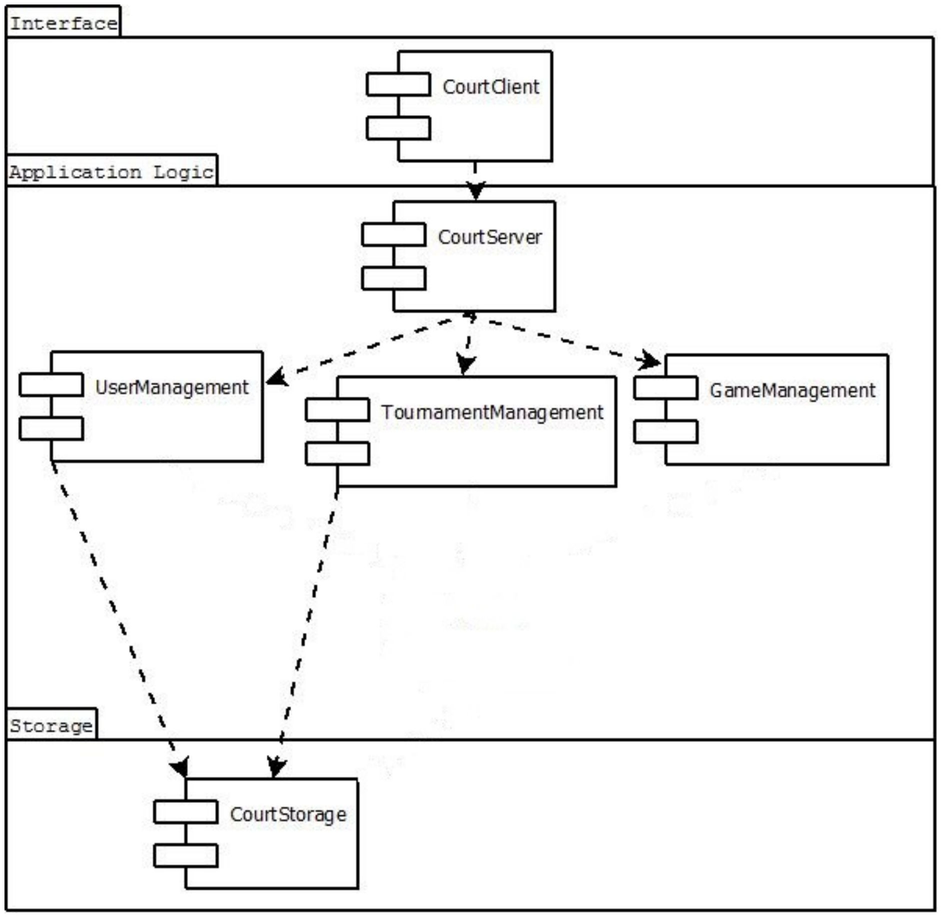
Currently it is only possible to play CSP games using the existing system, large parts of the code are hardwired to this game. The success of the current implementation has proofed that the general concept of a court is highly efficient and it is desirable to apply the same court games for other domains. This reimplementation has the goal to offer a more generic framework, designed with the idea of new game types already in mind.

The main architectural issue of the current implementation is, that the server becomes a single point of failure. The system (old as well as new) however depends on a central, trusted instance - this eliminates alternative approaches that would be more robust (Peer-to-Peer). This weakness has to be accepted, but of course can be alleviated by strongly focusing the development on reliability.

Another issue of the client-server approach is the fact that time-bound calculations that are executed on clients can vary depending on hard- and software of the connected client. The obvious solution to this problem would be to execute all the code on a central instance, this however would multiply the time required for each competition and would render the already long lasting tournaments unusable. To ensure fair tournaments, the participating players therefore have to agree on the hardware/software combination that is used for the clients.

3. Proposed software architecture

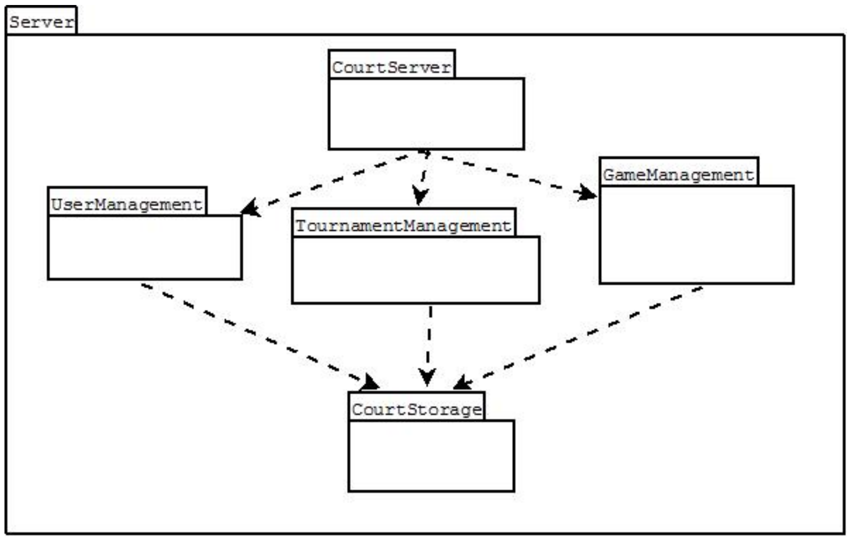
3.1. Overview



The architecture of the arena system is based on three layers, the topmost layer consists of the client as well as the client-connection parts. The middle layer realized that actual application logic and manages games and tournaments as well as their participants. The storage layer abstracts the components that are used for storage purposes (logging, results, etc.)

3.2. Subsystem decomposition

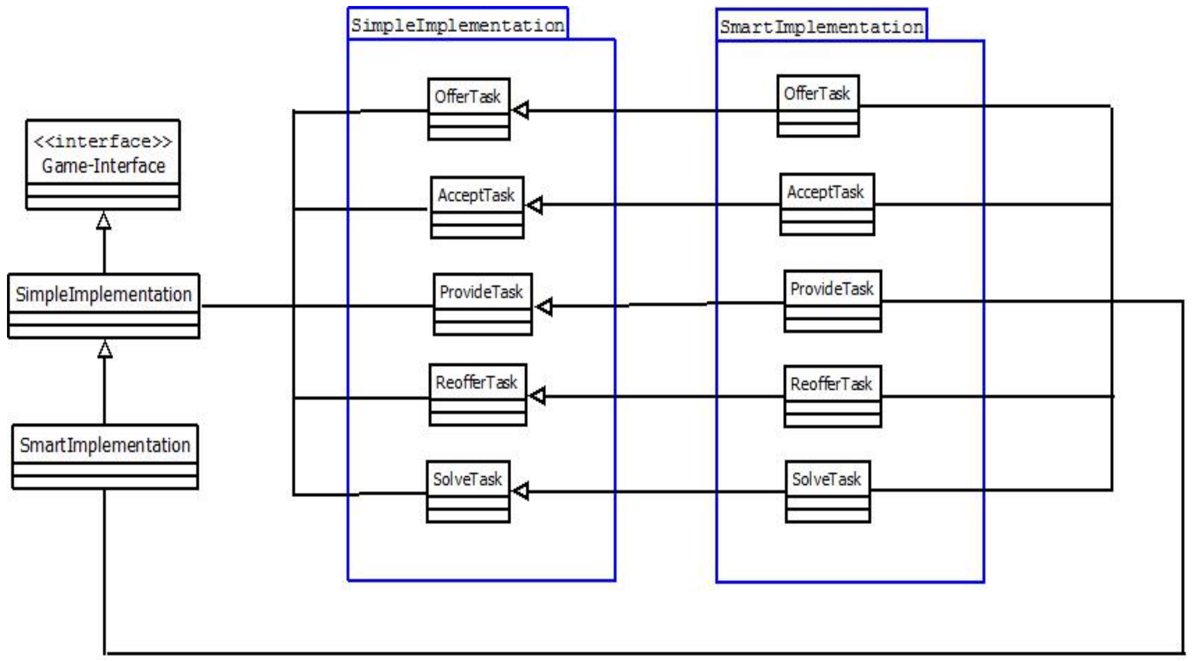
UML: Server



The server is divided into the following five main packages:

- CourtServer: Bootstraping of the server, configuration
- UserManagement: Registration and management of participating users
- TournamentManagement: Creation and management of tournaments
- GameManagement: Creation and management of games
- CourtStorage: Wrapper for the storage and logging components

UML: Client



The Implementation of the simple ('baby') avatar allows developers that use Java for the implementation of an Avatar to reuse the "generic" parts of an Avatar. The SimpleImplementation class delegates all required tasks to specific implementation classes. These take care of the required administrative tasks and provide random based dummy implementations of the actually interesting methods of a domain. Only these very specific tasks then have to be overwritten by the programmers.

3.3. Generation of the Domain Specific Elements

Grammar and Parsers / Printers

The current implementation already hints at a solution for a generic approach. Each specific game type is a combination of the generic, base components and the game specific classes. If both of these grammars are defined as DemeterF compatible 'Class Dictionaries' [DemF] the concrete source code as well as the required parsers and printers to exchange data can easily be generated. The specific parts correspond to the elements in the section 'Teminology'.

The following two sections show how such a division into a generic and a specific section of the grammar could look like. These class dictionaries can be used to directly generate Java and C# code and further allow anybody to write parsers and printers based on this specification. The combination of a language neutral grammar and the usage of HTTP opens up the possibility for implementations in any implementation language.

3.4. Baby Avatar Generation

The generation of a baby avatar for a new game type consists of the following two steps:

- Generating the Java classes based on the new specific CD as well as the generic CD
- "Baby" implementation of the required interface, since the data used for the game are well defined, the implementation of these methods could even be automated.

3.5. Server Package Generation

The generation of an additional package to provide server support for a new game type consists of three parts:

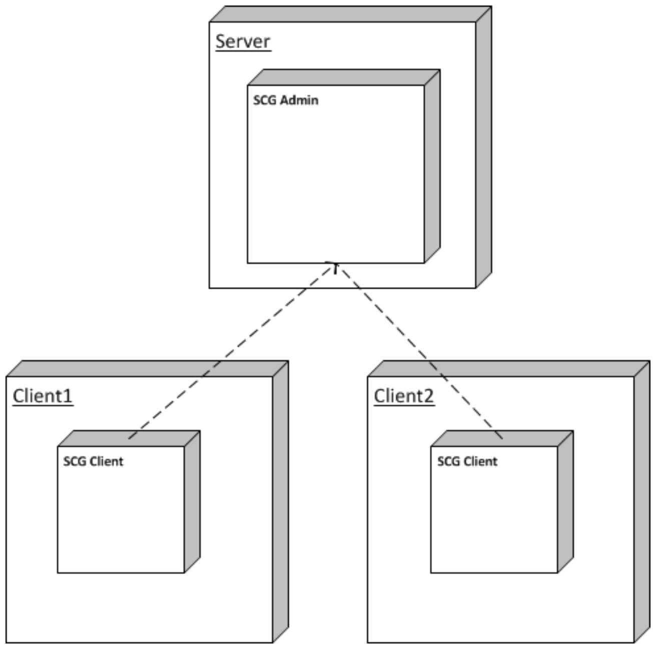
- Generating the Java classes based on the new specific CD
- Making these classes available to the JVM running the server
- Activating the new game type on the server

If the JVM supports dynamic class loading, these new games could be added without having to take the server offline.

3.6. Hardware/software mapping

The hardware and software of the server as well as the client play a secondary role, the system does not require a special environment except for the JVM as well as network access between each client and the server. There is no need for a direct connection between any client.

The network protocol that is used for the communication between client and server is HTTP. This decision has the advantage that most of the network specific details can be abstracted and the developers of client as well as server can focus on the domain specific data. Implementing the client-server data exchange on top of a open standard such as HTTP has further the advantage that any client could communicate with the server and participate in tournaments. The specific ports on the server as well as on the client side have to be white-listed on firewalls between client and server. If this cannot be achieved the competitions have to take place within Local Area Networks. To ensure fair competitions in games that are time-bound it is mandatory to standardize the hard- and software of the participating clients.



3.7. Persistent data management

The system features an all-purpose logging solution based on the widely used SLF4J [SLF4J] / log4j [LOG4J] packages. This combination allows the use of the fine grained logging API without having to worry about the actual implementation. In the early stages or for testing purposes a light-weight logging implementation can be used and it is no problem to scale up at a later stage to redirect the logging data to a database.

The statistical information collected during the games will be directly written into a data pool that allows an analysis of past games and tournaments. The code will only be coupled to a vendor neutral layer that translates the read / write commands to the specific solution.

To ensure that generated information can be stored in a way that is optimal for the individual type of data the new implementation of the SCG Court allows the implementation of game specific storage layers. These allow creators of new game types to store the results of competitions in optimal ways (special storage for large data, images, etc.)

3.8. Access control and security

Current Approach

Clients sign up for a tournament during a pre-registration phase. In this step the clients specify a password that is later used to verify that the connecting client is the same as the one that got pre-registered earlier. Only clients that have a valid combination of name and password are allowed in the tournament, there's no option for a late registration. The authentication information is only used when a client registers for a tournament, after this phase, every answer that comes from the registered IP / Port combination is accepted. This simple approach leads to another trade off:

Security v. Flexibility	The simple approach allows clients to be completely exchanged during the tournament, on the positive side, this allows a team that discovers a problem during a tournament to fix it, in a time-window where the client is not asked any questions. The same flexibility however also allows somebody to shut down a running player and replace it with some other implementation.

Possible Improvements

Encrypt the whole HTTP data transfer using SSL and Client-Certificates. This approach would ensure that only developers that are in possession of the private key could provide and possible manipulate a client that participates in a tournament.

3.9. Global Software Control

Generally all communication between server and client follows the 'Hollywood Principle' [HOLW]. In this concrete case, this means that all events during the lifetime of a tournament are triggered by the server and the clients are expected to respond accordingly. Messages coming form a client that has not been requested to send a specific message are ignored and can lead to the disqualification of the player. Responses from clients are verified by the server (for validity and possibly integrity, see 'Access Control and Security'). Depending on the severity of the problem in a message the player is either disconnected or granted another chance to respond according to the protocol.

The administrator/server will initialize each game of a tournament in the same fashion, following the protocol of the game.

3.10. Boundary conditions

The following section lists the regular as well as the exceptional cases for the lifetime of a tournament:

(Planned) Server start	The server starts and loads all the game types that are known so far. After this phase tournaments can be configured and started
(Planned) Server stop	The definition of a planned server stop is the following: Termination of the process after all tournaments are finished. None of the clients are bothered by this step, all results are stored.

Unplanned Server stop	If the server process is terminated for some reason, the results of the current round are lost. The server however keeps track of all Events and enables tournaments to be resumed starting at the round before the server was stopped.
Client connect (registration)	If the client connects and supplies the correct information, the client is listed as correct participant and will be queried as soon as the tournament is started.
Client disconnect (during tournament)	If a client correctly participates for some time, and then stops to respond, the client is immediately eliminated from the tournament. Even if the client recovers shortly after this disconnection it will not be allowed in this round anymore.
Client disconnect (before tournament start)	If a client correctly registers but does not respond to the first challenge, the client is immediately eliminated from the tournament. Even if the client recovers shortly after this disconnection it will not be allowed in this round anymore.
Client disconnect (after tournament end)	This is the regular case, the server is not bothered by this event.

4. References

[CSP] http://www.cis.temple.edu/~ingargio/cis587/readings/constraints.html
[SCG] Specker Challenge Game: Requirements & Design. From Generic to CNF/CSP Versions: http://www.ccs.neu.edu/home/lieber/courses/cs4500/f09/requirements/requirements-cs4500-f09.pdf

[SCGPG] http://www.ccs.neu.edu/home/lieber/courses/se-courses/cs5500/sp11/projects/playground-designer-user-guide.html

[LOG4J] http://logging.apache.org/log4j/1.2/

[SLF4J] http://www.slf4j.org/

[HOLW] http://c2.com/cgi/wiki?HollywoodPrinciple

[DemF] http://www.ccs.neu.edu/home/chadwick/demeterf/

5. Glossary

- Avatar - Participant in an SCG game that has the ability to generate claims for a specific domain as well as to react to given claims.
- CD - Class Dictionary
- CSP - Constraint Satisfaction Problem, see [CSP]
- SCG - Scientific Community Game