Genius 2.0 requirements

# Required core (new/improved) functionality

* Dynamic domains
  + New issues during a negotiation session
  + Change value range
  + Concerns:
    - Server security
    - What if someone else changes my domain?
    - What if someone else changes the domain that is being used in a negotiation session
  + Current approach: file based. domain/profile server reports to listeners if something changed. The listeners then re-load the domain and profile as they like..
* Dynamic profiles.
  + Works like dynamic domains: file based
* Add new protocols (offline, static)
  + Issue by issue negotiation
  + Majority voting e.g.
  + Complete package negotiation
  + Mediated / not mediated
  + Current approach: protocols are java files inside the server. Precompiled. 3rd party protocols are fully checked before incorporation by us.
* Add new types of domains and preference profiles, for example,
  + Linear
  + Non-linear
  + Incomplete profiles
  + Hierarchical preference profiles, including interests (tree structure). Currently we decided to drop hierarchical profiles, as Catholijn does not want to release this technology yet.
  + Qualitative profiles (fuzzy)
* Add new analytical tools
  + R, Rstudio (R might not be free)
  + Java script \* Nagoya students will take a look at :)
  + Some free libraries?
  + Python libraries?
  + Current plan: no analysis tool yet because we still need to decide under which conditions an agent shares its utilities (profiles can be kept private).
  + If user needs to see graphs, he uses an analysis tool. The runtime overview just provides the logs with the actual actions, and these actions do not contain the party’s utilities of the bids.
* Agents can negotiate
  + through <channel>
    - sockets (the connection has to be open)
    - Email (no special connection).
    - We need to discuss the e-mail plan. Actually e-mail is a VERY special connection. It is not clear what the idea behind this is. How we want to manage e-mail accounts and where Who will pay for that. etc.
  + Some object/file
    - JSON objects
    - XML file (current plan is to use exclusively JSON)
  + it ispossible for any agent to
    - Using any programming language (performance difference stem from the language)
    - Choose for Python and the python machine learning libraries
    - Use python from java to use python using java-python bridge as well
* Have user-friendly interfaces for
  + - Running a tournament or negotiation session
    - Analysing a tournament or negotiation session
* Have exercises for students (what does this have to do with core functionality?)
* Have templates of python and java agents to help students and new researchers to build their own agents
  + - Following the BOA
    - Additionally, the new BLA structure (what is this?)
    - For learning from past negotiations
* Tournament/Competition
* Agents written in different languages (but we support a few, eg java and python)
* Protocol management (this is done on server)

# Future Requirements

* Preference elicitation GUI Jaafor nonlinear utility space

# Coding team

* Cihan Eran - [cihan.eran@ozu.edu.tr](mailto:cihan.eran@ozu.edu.tr)
* Wouter (Delft), [w.pasman@tudelft.nl](mailto:w.pasman@tudelft.nl)
* Fukui Tomoya [fukui.tomoya@itolab.nitech.ac.jp](mailto:fukui.tomoya@itolab.nitech.ac.jp)
* Toyama Tatsuya [toyama.tatsuya@itolab.nitech.ac.jp](mailto:toyama.tatsuya@itolab.nitech.ac.jp)

# Maintenance & Architecture

* Support through maven or equivalent dependency management.
* Modular setup so that you can import only the needed functionality in other projects.
* Core functionality should be implemented such that there are no limits to the size of the domain (eg, do not try to iterate over the bidspace)
* Client/server setup
  + Motivation: People can write their agents in whatever language they like, as long as they satisfy the protocol it is fine.
  + Run competition as usual? We have a choice here: we still run it locally (people submit their agents) or the agents remain at the client side (i.e., on the participant’s side). In both cases holds that we can’t check all agents in all possible languages, so we don’t check for cheaters in competitions (which was difficult to start with). We count on fair play.
    - Locally at tournament site:
      * controlling the amount of time that agents get is impossible. However, when time is up, you just kill the agent process. Of course such a session would not end in a negotiation result, which will in the course of the tournament mean that that agent loses the competition.
      * We use the PartyFactory system to start up required agents automatically as necessary.
    - Distributed: tournament manager at tournament site, and participants at participants’ sites:
      * a participant could choose to run a supercomputer and so win the competition.
      * Each participant can use all available negotiation time for its computations.
      * We have almost no possibility to check on cheating. We can still demand to get their agent code, and test if it runs on our machines.
      * Each participant should start his/her agent in time and has to make sure it keeps on running. This might turn out to be problematic.
  + Current approach: all distributed: domainserver, partyfactory server and run server. The servers CANactually be on same machine (speeds up all network communications). Not clear what “cheating” means now, we have a direct network connection with each agent during a run and as long as we receive messages approved by the protocol it’s fine.
* The Session/Tournament runner will be set up as a listenable runner. Therefore the server will be “listenable”. To handle callbacks to the listener in web based apps, we need to use WebSockets as web apps do not allow other kind of sockets to connect with. WebSocket is new and relatively insecure. The least we should do is use the wss (more secure) protocol and try to follow security guidelines eg https://devcenter.heroku.com/articles/websocket-security.

What is this about?

* + Old Code pieces
    - Core
    - Agents
    - Domains
    - NegotiatorGUI
  + Nieuwe packages (niet projecten!):
    - Definitions : domains, agents, protocols, profiles
    - Repositories of agents, profiles, domains, protocols
    - Elke agent, profile, etc verwijst naar definitions
    - Core verwijst naar definitions en kan agents aanroepen
    - GUI verwijst naar definition en kan core en agents aanroepen

# [Uses cases](https://docs.google.com/document/d/1AWmP2dmNV6LHbo9dg0ZtJg7G-Amf0fdcdjop-3gJ4MQ/edit)

* Competition
  + Local variant
  + Distributed variant
* Development of new protocols, strategies, agents
* Running a negotiation with agents on different machines (including top ‘old’ agents)
* Demo mode (light, offline)
* Other applications that use Genius functionality, e.g., PN

1. JVM supported languages
   1. Can they reach/use java classes (that would enable us to share parsed objects)
   2. How to be in control of these agents?
2. non-supported by JVM
   1. Socket approach to exchange information
   2. bids, protocols, profiles, domains, anything else?>
   3. How to manage protocol and stay in control (for fairness)
   4. Security
   5. Use / develop another protocol for fair time division?

Catholijn suggereert dat nieuwe agents een tournament/sessie in kunnen schuiven. Misschien voordat de sessie start? Of misschien ook halverwege?

GeniusWeb is volgens Tim niet in bruikbare vorm: houtje-touwtje, veel bugs.