Justified collective decision-making in agri-food systems: an approach based on computational social choice, argumentation methods and tools (Supplementary Material)

Appendix 1: Rank sliding approach example

By applying our rank sliding approach on Example 3 we can get:

- *Global Warming: both Alice's and Bob's rank for the globalWarming alternative have accepted justification, therefore they are not changed. However, Carol did not justify her ranking and thus her justification is rejected. Her rank of 3rd for globalWarming should be changed to the closest rank with accepted justification, which is 2nd;*
- *Water Consumption: Bob's rank for water consumption has an accepted justification therefore it is not changed. Alice's rank for water consumption has an ambiguous justification while Carol's rank has a rejected one, thus they are both changed to the closest rank with accepted justification, which is 3rd;*
- *Land Use: both Alice's and Bob's rank for land use have an ambiguous justification, and since there is no rank with an accepted justification, they remain unchanged. Carol's rank has a rejected justification therefore it is changed to the closest admissible rank, which is 2nd (Figure 1).*

The final updated rankings are:

- ***Alice:** globalWarming (1) > landUse (0) ~ waterConsumption (0).*
- ***Bob:** globalWarming (2) > landUse (1) > waterConsumption (0).*
- ***Carol:** globalWarming (1) ~ landUse (1) > waterConsumption (0).*

Using the score voting method, we can find that globalWarming has a score of 4, landUse has a score of 2, and waterConsumption has a score of 0, therefore the final aggregated ranking is: ***globalWarming (4) > landUse (2) > waterConsumption (0)****.*

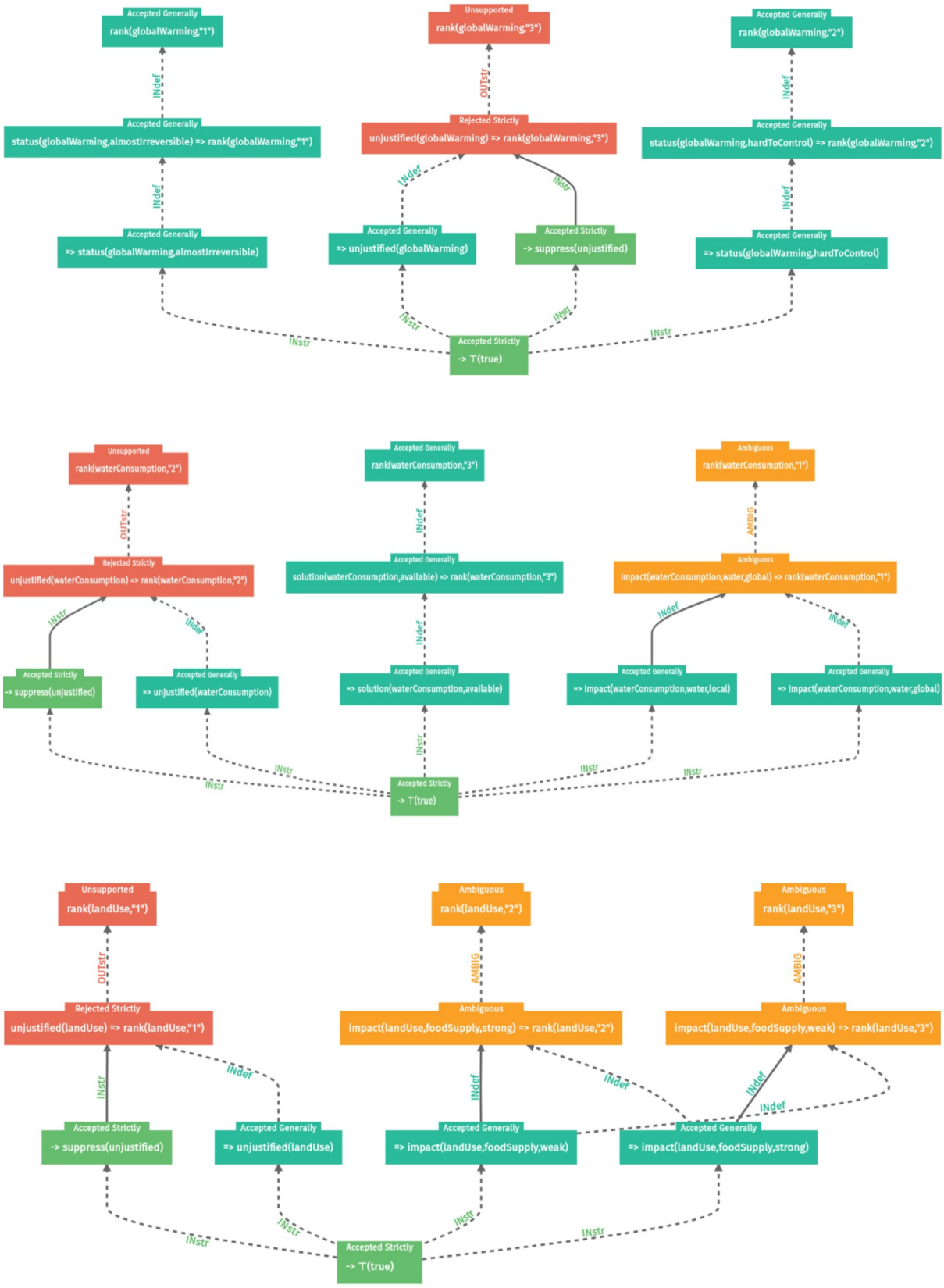

Figure 1. *Statement Graphs for the globalWarming (top), waterConsumption (middle) and landUse (bottom) ranking and justifications.*

Appendix 2: Established vocabulary for justification representation

We have established the following vocabulary to represent explicit and implicit knowledge expressed in the survey:

- "**impact(Category,Something,Type,Strength,Level)**" to express the *strength* (strong, medium, weak) and *type* (positive, neutral, negative) of the impact that an impact *category* has on *something* (e.g. freshwater, environment, etc.) at a specific *level* (local, global, etc.). For example, "impact(landUse, foodSupply, negative, weak, global)" which translates to "landUse has a global weak negative impact on foodSupply".

- "**accordingTo(Source,Something,Adjective)**" to express that a Source (e.g. media, own knowledge, etc.) has qualified Something (e.g. landUse) by an Adjective (e.g. important, etc.). For example, "accordingTo(media, landUse, veryImportant)" – "According to media, landUse is veryImportant".
- "**solution(Problem,Type,Adjective)**" to express that there is a Adjective (e.g. available, easy, etc.) solution to the Problem (e.g. deforestation, etc.) with a specific Type (e.g. political, technological, etc.). For example, "solution(freshwaterEcotoxicity,technological,available)" – "A technological solution to the freshwaterEcotoxicity problem is available)".
- "**action(Action,Adjective)**" to give an action an adjective. For example, "action(increasingLandConsumption,unsustainable)" – "*increasingLandConsumption* is *unsustainable*".
- "**essentialFor(Something1,Something2)**" to express the strong link between certain concepts. For example, "essentialFor(land,foodProduction)" – "land is essential for foodProduction".
- "**relatedTo(Category,Something)**" to express the different concepts that are related to an impact category. For example, "relatedTo(landUse,deforestation)" – "*landUse* is related to *deforestation*".
- "**cause(Something1,Something2)**" to express that Something1 causes Something2. For example, "cause(populationGrowth,increasedDemandForWater)" – "populationGrowth causes increasedDemandForWater".
- "**status(Something,Adjective,Level)**" to express the state of Something with an Adjective and a Level (global, local, etc.). For example, "status(freshwater,rare,global)" – "the status of freshwater is rare on a global level".
- Other case-specific predicates have been added depending on the needs, such as "**sameAs**", "**moreImportantThan**", "**lessImportantThan**", etc.

Appendix 3: Description of tools' modules

3.1 The DAMN tool

The **Knowledge input, display, and collaboration module** has been implemented as a stand-alone tool called DAMN (Defeasible Reasoning With Statement Graph) available at the following link: https://ico.iate.inra.fr/damn/. The role of this module is to collect the knowledge of the agents, display the Statement Graph, and allow these agents to collaborate in real time by updating their knowledge base. It is the *"entry point"* of our decision making platform, therefore it interacts with the agents (or a data engineer representing the agents) and exports the knowledge bases of each agent as a JSON (JavaScript Object Notation) file (a lightweight data-interchange format that is widely used on the web) while also displaying the resulting Statement Graph produced by the reasoning module.

In the DAMN tool, every decision problem is called a "project" (Figure 2). The project can have multiple agents, each agent has his own knowledge base. A key difference must be made between an *"agent"* and a *"user"*. An agent is an entity that takes part in a decision making problem, it has its own knowledge base and can represent a person or a group of persons. A user is a person that interacts with the tool, it can handle different agents or represent one single agent.

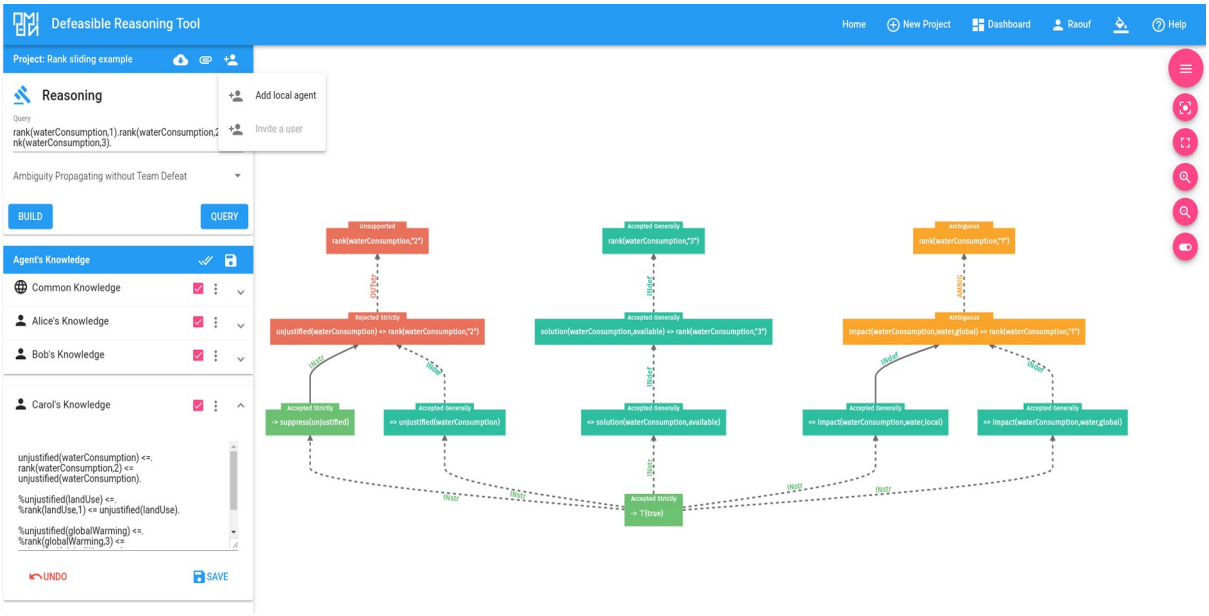

Figure 2. Project webpage of the DAMN tool.

The typical workflow of the tool can be described as follows (Figure 3): (1) A user logs in or creates his account on the platform; (2) he can then create or open a previously created project; (3) The user can add or remove agents to the project along with their knowledge bases that can be imported from a DLGP file (a format used to describe rules and facts); (4) He can invite other users to collaborate, insert, and modify the knowledge of one or multiple

agents in real time. (5) The user can build the Statement Graph from some or all the knowledge bases and check the status of a justification by inputting a query and choosing a semantics; (6) He can then interact with the resulting displayed graph.

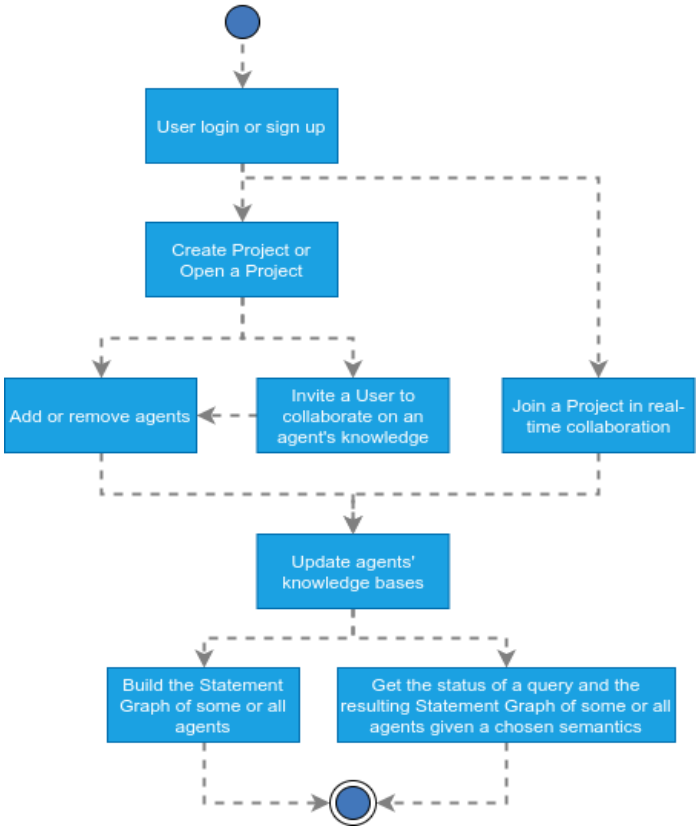

Figure 3. A typical workflow of the DAMN tool.

Concerning the tool's implementation, DAMN is built using the client-server architecture where the **front-end** describes the interface that the user will interact with and the **back-end** describes the different servers that handle data storage, collaboration with other agents, and interaction with other modules (Figure 4). The source code of the tool is available at https://github.com/hamhec/dam

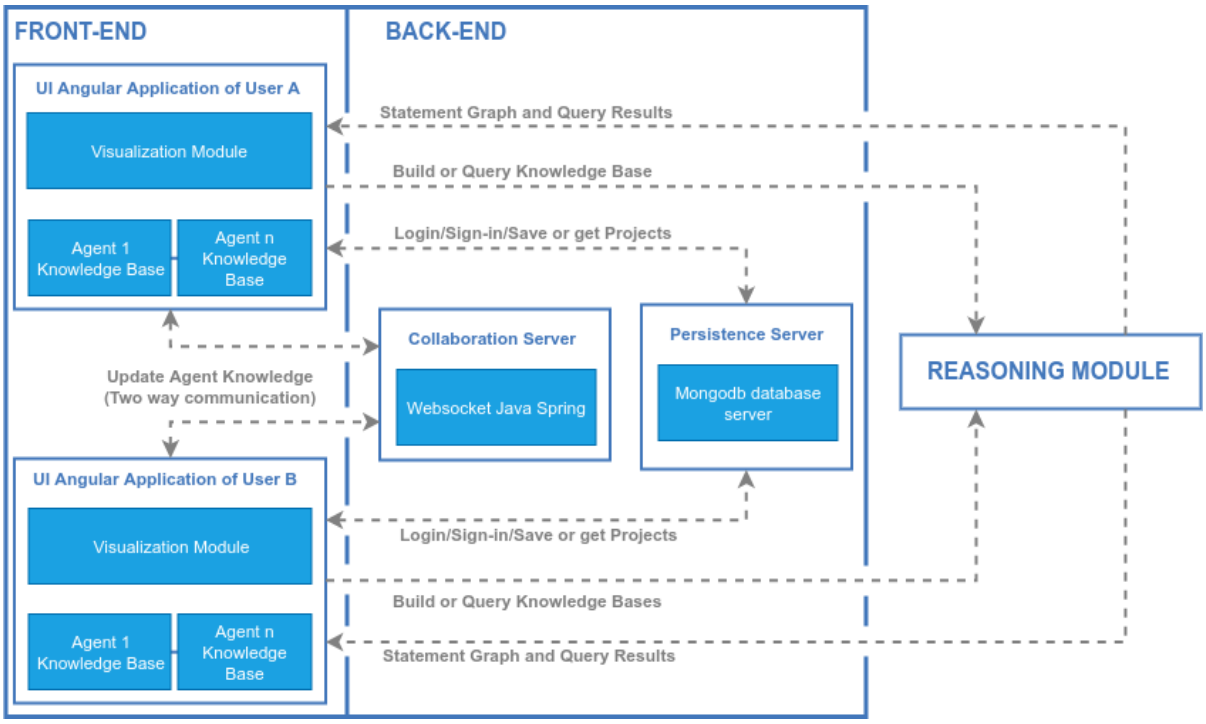

Figure 4. DAMN tool architecture.

3.2 The ELDR tool

The **Reasoning module** has been implemented as a stand-alone tool called ELDR (Existential Logic for Defeasible Reasoning). The tool is packaged as an API (Application Programming Interface) and is available as a JAVA API module in the official public repository (https://search.maven.org/artifact/fr.lirmm.graphik/graal-elder/1.0.17/jar).

The role of this module is to build the support and attack links of the Statement Graph and give the status of the justifications (accepted, rejected, or ambiguous) depending on the chosen semantics. It is the *"brain"* of our decision making platform and takes as input the JSON file of the agents' knowledge bases and outputs the Statement Graph with or without the status of each reasoning step as a JSON object. The user can directly import this module from the API repository and use its functionality.

The implementation of the ELDR tool is a JAVA API uploaded on a REST server (Figure 5). This means that it is accessed not through a graphical interface but rather through the ability to call a set of functions on the web and obtain the resulting Statement Graph as a JSON file. The source code is available at the following link: https://github.com/raouf2ouf/graal-elder.

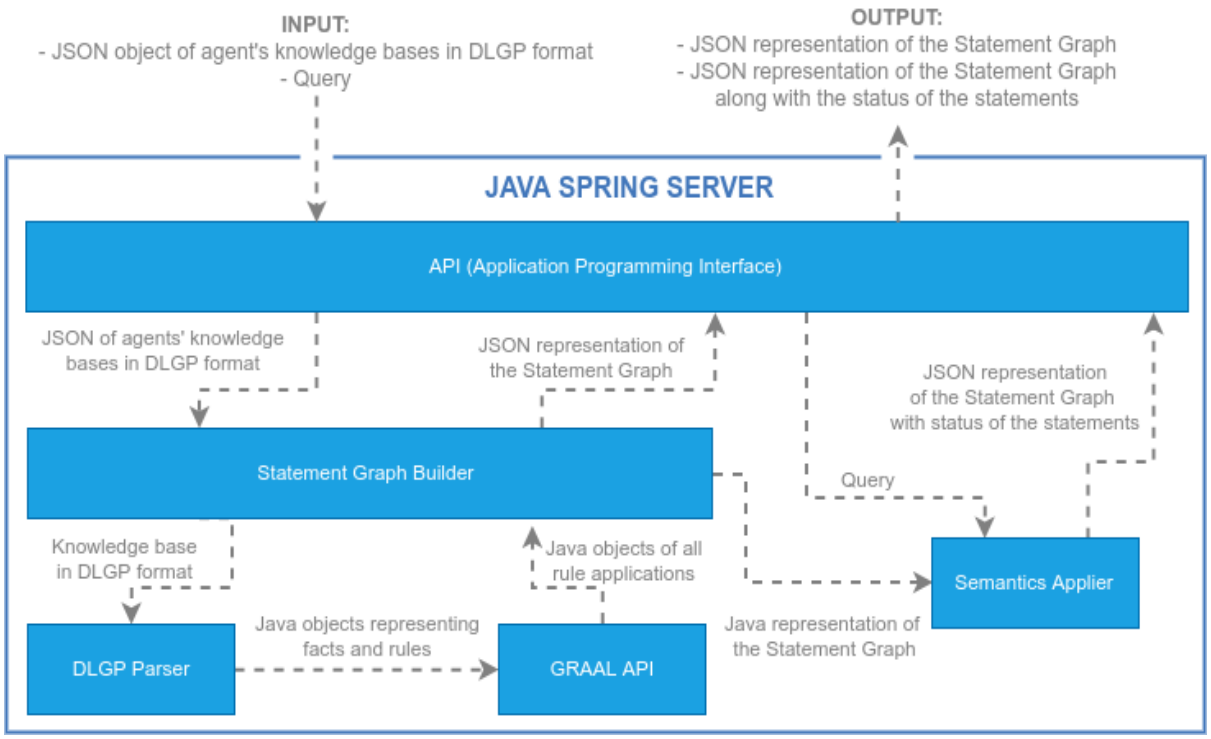

Figure 5. The ELDR tool architecture.

3.3 Rank sliding approach

The **Justified rankings module** is a direct implementation of our rank sliding approach. The module is packaged as an API composed of two parts: a function that exposes the implementation of our rank sliding approach, and a set of abstract functions that can be used as a basis for future implementation of different rank changing approaches.

The role of this module is to update the rankings of each agent depending on the status of their justification. It represents the “decision making” part of our decision making platform. It takes as an input the Statement Graph with the status of each reasoning step and exports an updated justified ranking for each agent.

The implementation of the module has been included in the DAMN tool. The module exposes a single JAVA class with a function which is based on the following input and output. The input is a JSON object that represents the Statement Graph with the status of each justification and the knowledge bases of each agent. The output is a JSON object that gives for each agent the new ranking of the alternatives.