HUMAN-MACHINE INTERFACE

AGI INTROSPECTION IMPLEMENTATION

Mykola Rabchevskiy
Jun 18

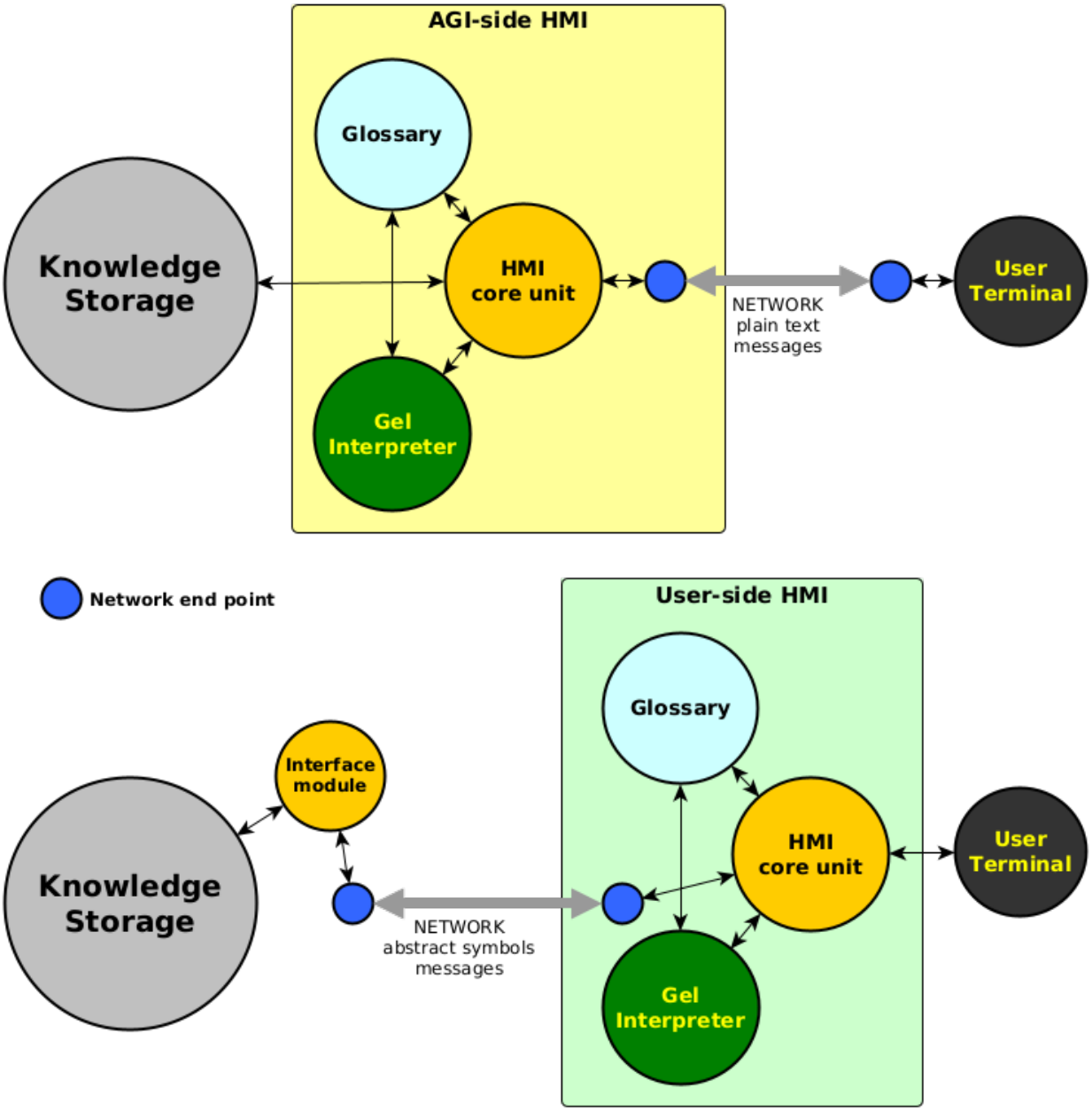
As noted earlier, the introspective capabilities of AGI are essential since they underlie the interpretability and correctability of the behavior of the AGI system. In turn, the effectiveness of the implementation of introspection significantly affects the overall assessment of the implementation of the AGI system.

Our version of introspection is based on the use of the communication language **Gel**, the lightweight version of which the **Knowledge Storage** modules make up the test application, which we describe in this chapter. The program allows to practically evaluate both the **Gel** language itself and the **Knowledge Storage** modules:

https://github.com/mrabchevskiy/Gel

Both the **Knowledge Storage** modules (in this case, **Semantic Storage** and **Data Storage**) and the **Gel** language claim the generality required from AGI; the consequence of this is a large set of knowledge manipulation functions and the corresponding structures of the communication language. This chapter provides information on installing a test application and performing basic queries to **Knowledge Storage**; in subsequent chapters, we will discuss functionality and language constructs.

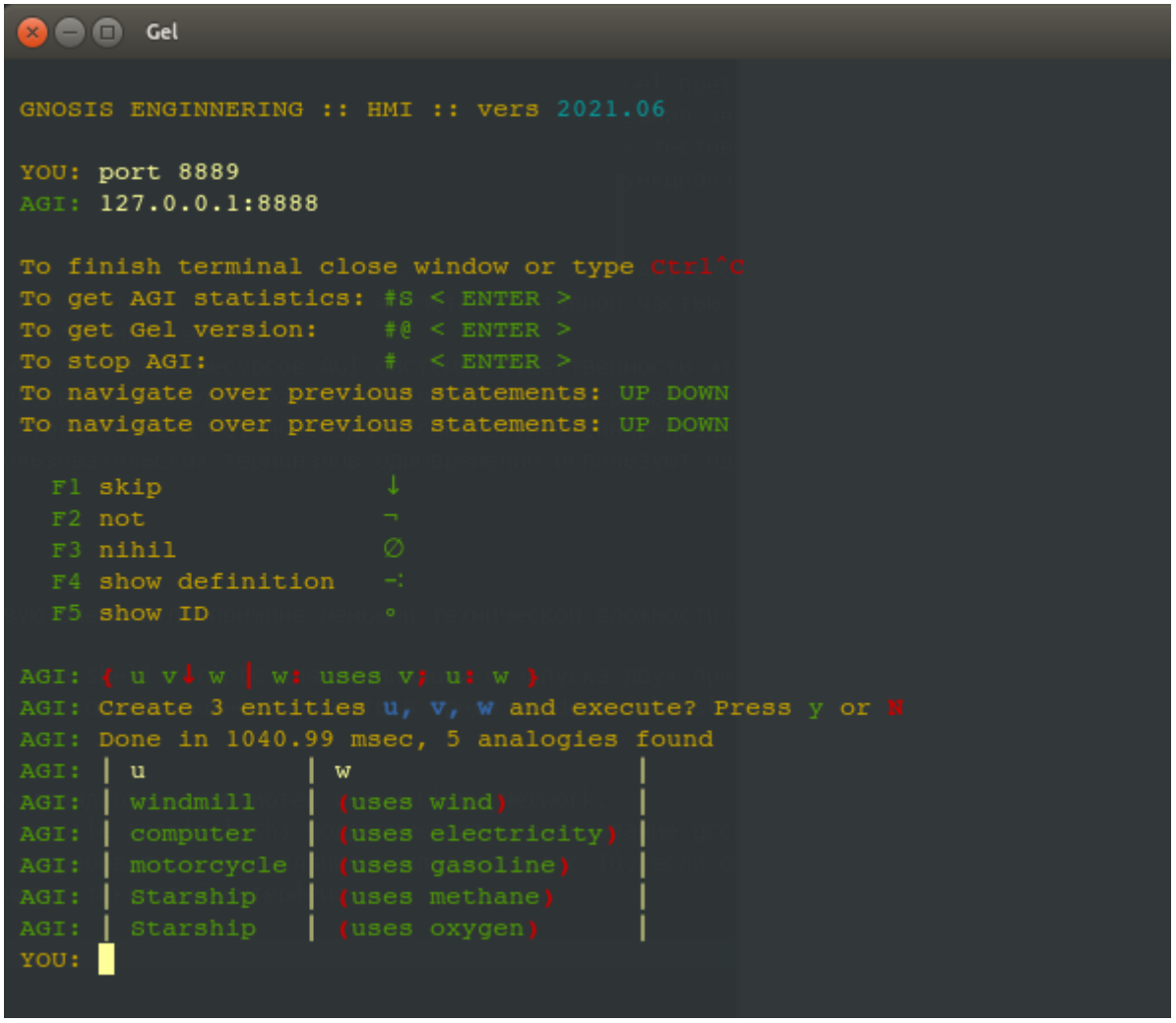
simultaneously use the same human language.



In this case, the first option is implemented (primarily due to the lower technical complexity).

The Github repository contains both C ++ source code and shell scripts for compiling and running two applications and ready-to-use executables for 64 bit Linux ('**hmi**' and '**terminal**') that can run in **Windows Subsystem for Linux** (**WSL**) environment.

The source code has no external dependencies other than the standard C ++ and Linux Network libraries. Shell scripts for compilation (**compile-hmi.sh** and **compile-terminal.sh**) assume GCC-10 and the C ++ - 20 standards; if used in a **WSL** environment, the GCC-10 compiler may need to be installed if it has not been installed before. Both applications have a text interface (log is displayed in the AGI application window):



Semantic Storage and **Data Storage** of the test application already contains the information provided in SEMANTIC STORAGE.

Preloaded test data allows executing queries without first entering any information. For example, to get a list of signs of the entity ***methane***, you should run the query:

methane ?

To get a list of entities with the ***methane*** sign, execute a statement:

? ***methane***

After pressing ***ENTER***, the text of the **Gel** statement will be sent from the console to the main application, where the statement is analyzed and returned to the user terminal in a 'colored' form, where, if there are no complaints about the syntax on the part of the interpreter, the entities known to the AGI system are highlighted in green, and the unknown ones are highlighted in yellow. If the user allows execution, AGI executes the statement, starting with the addition of previously unknown entities to **Storage** and the dictionary. Thus, ***new entities are added as soon as they are mentioned in the Gel statement***.

A two-step validation process that looks a little annoying at first glance avoids creating unwanted new entities due to typos in the entity names.

If the statement contains an error, then the coloring shows the location of the error, and instead of asking for permission to execute, an explanation is given.

Storage also contains a set of ***congenital*** concepts that are used as attributes of entities.

They are protected from modification and deletion by ***IMMUTABLE*** and ***IMMORTAL*** signs, which can be assigned to newly created entities. Due to the presence of these features, the list of congenital concepts can be obtained by executing the command:

? ***IMMORTAL***

The names of all congenital concepts are in the upper case to reduce the risk of naming conflicts.

Let's start with the simplest operators. Getting the complete definition of a Starfish entity:

Starship **–:**

Creating a new entity 'ant':

ant

Adding sign ***animal*** (note dot at the end):

ant: animal .

Sign exclusion:

ant: animal .

Addition and exclusion of several signs with one operator:

ant: animal ¬machine .

Forgetting an entity:

∅ ***ant***

List of the preloaded test entities to experiment with:

hydrocarbon
computer
electricity
gasoline
ground
methane
motorcycle
oxygen
produces
reusable
space
Starship
uses
vehicle
wind
windmill

In the following chapters, the Gel operators will be discussed in detail.

Keep in touch with us!

← Previous Next →