Declarative Semiformal Language for A.I. Assisted Software Development

oZumbiAnalitico

The objective of this semi-formal declarative language is to enable developers to utilize Artificial Intelligence for creating applications without the need to directly write code. However, it ensures that developers retain control over how the program functions whenever necessary.

```
[Notepad]
1. Create a Notepad
```

Above is a simple declaration with absolutely no control of the output of the A.I.

```
[Notepad]
1. Creating Window ~ Create Menu Bar | Create a Multiline Edit Control
-> Creating Window ~ Create Menu Bar || Create load button | Create save button
```

The second example is a refinement of the declaration, instructing the A.I. to create a menu bar with load and save buttons, as well as a multiline edit control for receiving text input.

A declarative language is a programming paradigm that specifies what the computer should do rather than how to do it. With the advent of Artificial Intelligence, simply declaring what we want for an application should suffice, and this can often be done informally.

However, to maintain more control over the desired application, one should have the ability to specify their requirements with any desired level of precision. That said, developers should be aware that an A.I.-assisted application may have errors or undesired behaviors, inherent to the nature of A.I. Developers must clearly understand the program they aim to create and possess sufficient knowledge to identify and address errors or unwanted outcomes. The more knowledge a developer has, the better prepared they will be.

```
[ Simple Notepad ]{ Visual Studio, windows.h, C++ }
1. Initializing ~ Main () || Create the Main Window () | & Message Loop
-> Initializing ~ Main () || Create the Main Window () || Create a Menu Bar ()
| Create an Edit Control () || The Edit Control Must Occupy All the Client Area
-> Initializing ~ Main () || Create the Main Window () || Create a Menu Bar () ||
Create the "Save" item | Create the "Load" item
2. Menu Save Click ~ Save () || % exists a filename ||
save the content of edit control to a file with the filename
-> Menu Save Click ~ Save () || % exists a filename | % else ||
open a popup to save the file manually | register the name of the file.
3. Menu Load Click ~ Load () || open a popup to load the file | register the name of the file.

This is a semiformal declarative language, although there is symbols the
names and descriptions must be interpreted and adapted to create a functional application.
```

```
1. N. with N as a number, or ->, don't have any meaning, this is just an human annotation.
2. [ Name or Application Description ]
3. { Context of Application }
4. X ~ Y, means that X is an event and Y is a declarative expression of how the event should
be processed.
5. X || Y means that Y is inside the structure of X, is in his scope.
6. X | Y means that X and Y stands in the same scope (structure) and Y is performed after X.
7. & X, means that X is an repetition structure.
8. X (), denote that X is a function.
9. % X, denote that X belongs to an conditional structure (if-elseif-else).

Explains step-by-step the creation of the application.
```

The example above should be entered into any chat A.I. capable of generating code. It has been tested on Grok and Copilot. To compile, you need to install Visual Studio with the C++ development kit and create an empty window application project. Simply delete any unwanted files from the Solution Explorer (side bar) and replace the main application code with the output generated by the A.I., removing any previously existing code. Pressing F5 will execute the program, creating a simple Notepad application.

The A.I. even explains what the prompt accomplishes, so I don't need to elaborate further. The first part involves describing the program's declaration using semi-formal language. The second part is essential for both the A.I. and the user to understand the meaning of the declaration.

Alternatively, one could simply prompt, 'Please, A.I., create a simple Notepad.' However, the developer cannot accurately predict how the A.I. will respond. The declarative language allows specifying with a variable degree of precision what the A.I. should do. Moreover, the user retains a conceptual model of the application project and can constructively and easily modify it.

```
[ File Map ]{c++ console application, CLI, c++17 standart}
1. Initializing ~ $ string vector "files" | main() || & main loop || display message () |
process input ()
-> Initializing ~ $ string vector "files" | main() || & main loop || display message () ||
display the line message "[ File Map ]" |
display the line message "1. exit : exit the application" |
display the line message "white a file extension beginning with '.'"
2. User Input ~ process input () || % "exit" || exit the application
-> User Input ~ process input () || % "exit" | % else "show" ||
% files size is greater than 0 || show the lines of the string vector "files"
-> User Input ~ process input () || % "exit" | % else "show" |
% else string beginning with "." || % length greater than 1 ||
& search for directories and subdirectories || % file match the desired extension ||
put the complete filename on a string vector "files"

This is a semiformal declarative language, although there is symbols the
names and descriptions must be interpreted and adapted to create a functional application.
1. N. with N as a number, or ->, don't have any meaning, this is just an human annotation.
2. [ Name or Application Description ]
3. { Context of Application }
4. X ~ Y, means that X is an event and Y is a declarative expression of how the event should
be processed.
```

```
5. X || Y means that Y is inside the structure of X, is in his scope.
6. X | Y means that X and Y stands in the same scope (structure) and Y is performed after X.
7. & X, means that X is an repetition structure.
8. X (), denote that X is a function.
9. % X, denote that X belongs to an conditional structure (if-elseif-else).
10. $ X, denotes that X is a declaration for a construction of a variable.

Explains step-by-step the creation of the application.
```

This second application is simpler: it scans the current directory and subdirectories to find files with a specified extension. To create it, simply prompt any capable A.I., copy the generated code into an empty console application project, set the project to standard C++17, then compile and build the application.

Source code and manual compilation are becoming things of the past. With this approach, applications are constructed directly from their logical structure. This semi-formalism focuses on building the logic behind an application.

These two examples stem from imperative paradigm development. However, it is straightforward to introduce symbols that adapt the language to incorporate modularity, object-oriented programming, and other paradigms.

This semi-formalism paradigm is useful for:

1. Modeling the Application Logic
2. Project Refactoring
3. Project Expansion and Maintenance
4. A.I. Prompt Engineering for Software Development
5. Project's Logic Design and Development

The last example is too extensive to include in this document, but it can be accessed on GitHub (https://github.com/EYO-07/DSL). Essentially, the project involves creating a resource monitor using Windows Forms and the OpenHardware library. While the project was successfully developed with Grok AI's assistance, some fixes and adjustments to the original instructions were required. Therefore, a solid understanding of application development is essential to identify undesired behaviors and resolve errors.

The A.I. has attained a notable degree of consciousness—a logical and rational type of consciousness. While it still lacks emotions and desires, its ability to comprehend logic and language surpasses that of any human. If we consider 'Emotion, Intellect, and Will' as dimensions in the Euclidean space of consciousness, the A.I. represents a strong vector in the direction of intellect, compared to a smaller, more balanced human vector with components distributed across all dimensions.