Entologic

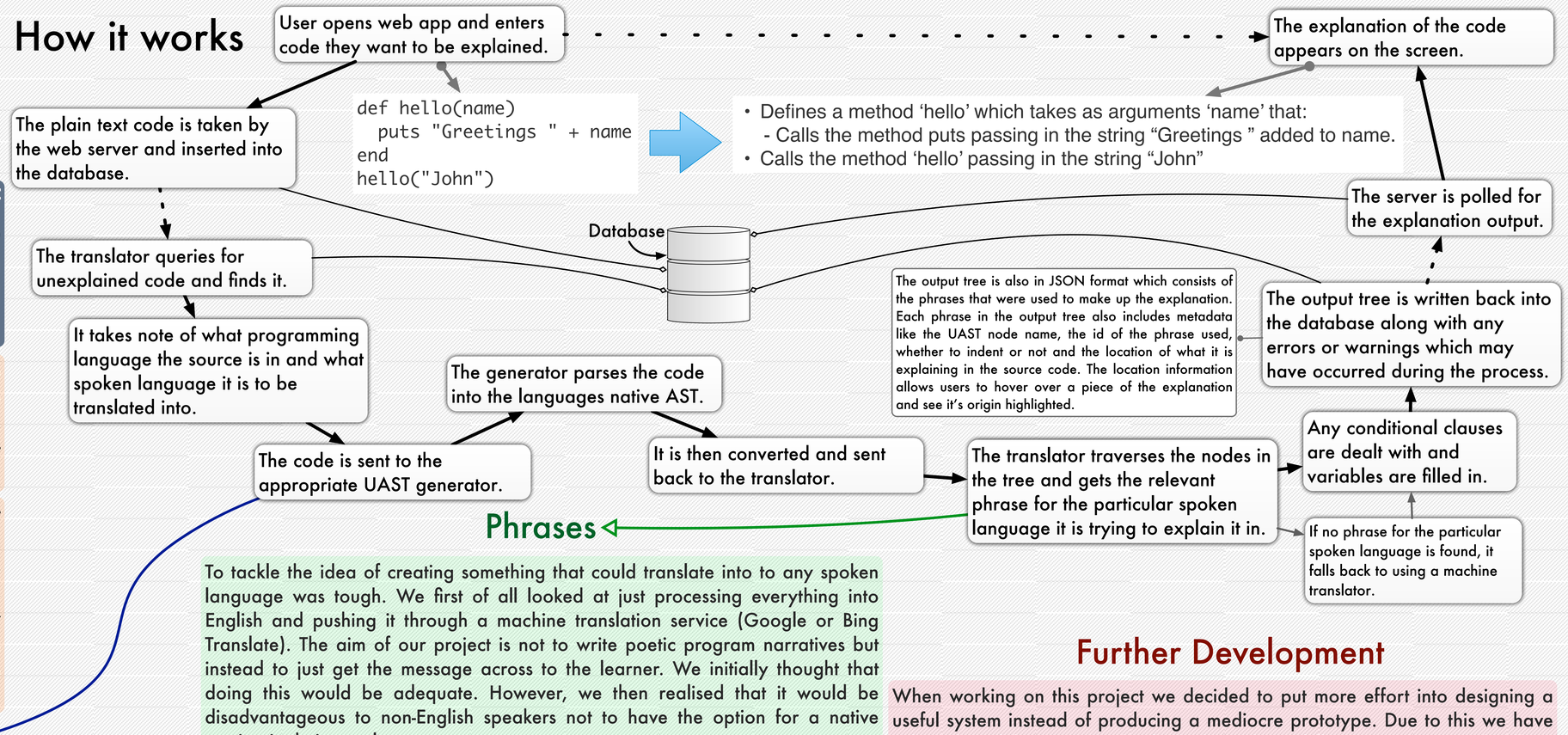
From the Greek words εντολη (instruction) and λογος (meaning).

Our project allows for automatic explanation of computer code from any programming language into any spoken language. This helps self learners have a better understanding of programming and in turn could improve the IT workforce in both size and capability.

While one can go to college to study computer science, learning to program shouldn't require a third level education. People frequently attempt to learn at home using the wealth of free tutorials and documentation online but large amounts of code without thorough explanation can be overwhelming.

We have attempted to solve this problem by designing a system that automatically explains provided code to learners piece by piece. Instead of forcing them to rely on merely copying patterns, it gives them a deeper understanding of what is actually going on, which is vital as a software developer. It can also be used in computer education as a learning tool for students and teacher alike. It even has its place in business if, for example, a non-developer wanted to know what a certain piece of the company software does.

How it works



The UAST

Before any program is run on a computer, the plain text instructions have to be represented in machine code (binary) so the processor can understand them. This process is called compilation. During this process the program is converted to something called an Abstract Syntax Tree (AST). This tree is a representation of the syntax in a programmatic form. It is here where we intervene to extract meaning.

One problem we came across when designing it was that every programming language has it's own format of AST. We would then have to write a translator for every language. To overcome this we created our own tree: the Universal Abstract Syntax Tree (UAST). In the specification we have added representations of all basic programming structures.

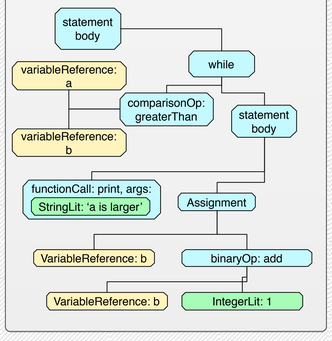
The UAST itself is formatted in JSON as it is a popular format supported by many programming languages. The system supports UAST generators for each specific language. We have released a specification for the UAST online which allows programming enthusiasts to create generators for languages they want to see integrated with EntoLogic.

Abstract Syntax Tree Example

An abstract syntax tree is the syntax of the code in a programmatic form. To give you an idea of what it is, here is a pseudo-code while loop:

```
while (a > b) {
    print("a is larger");
    b = b + 1;
}
```

Below is a visual representation of what the program would look like in an AST form:



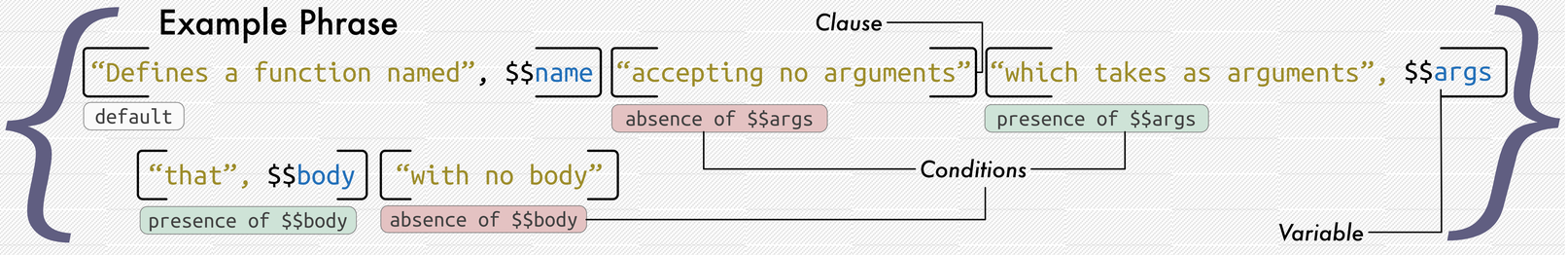
Phrases

To tackle the idea of creating something that could translate into to any spoken language was tough. We first of all looked at just processing everything into English and pushing it through a machine translation service (Google or Bing Translate). The aim of our project is not to write poetic program narratives but instead to just get the message across to the learner. We initially thought that doing this would be adequate. However, we then realised that it would be disadvantageous to non-English speakers not to have the option for a native version in their own language.

EntoLogic is designed to hold multiple 'phrases' for a particular part of a program. Each phrase is made up of multiple clauses (as shown below) which are conditionally controlled by what they are describing. Each clause can contain text and variables that are filled in depending on what was inputed by the user. While being specific to each spoken language, phrases can also be programming language specific as many programming languages have specific vocabulary for describing the same things. E.g. A 'Hash' in Ruby is akin to a 'Dictionary' in Python.

Similar to opening up the UAST to the community, we decided to allow for crowd-sourced creation of phrases directly on the web app itself. Whenever there is no specific phrase available, we fallback to using a machine translation service. If a user notices an inferior translation for a specific phrase, they can add their own one. Other users may agree and thus up-vote it. The translator will then use the highest voted phrase next time someone requests an explanation providing a more accurate adaptation.

Example Phrase



*A function is a programming structure which contains code in its 'body' and can be run later handing it pieces of data called 'arguments'. The function can use them to perform some operation and 'return' a result.

Further Development

When working on this project we decided to put more effort into designing a useful system instead of producing a mediocre prototype. Due to this we have a long list of features, each of which we have fully mapped out and will go towards helping the learner get a better understanding. Some of them include:

- Allow contributors to add phrases for core and standard library classes and methods which get placed inline with the description of the syntax.
- Analyse which phrases are being used the most through machine translation and suggest that they be worked on next by contributors.
- Contextualise function calls by filling in the variables being passed to them making it easier to concentrate on what else is going on in the body.
- Integrate with other online programming tools.
- Have commenting/discussion underneath each phrase where other users can point out suggestions in the creators attempt.
- Possibly monetise with methods including advertising on the site and prioritising explanation of paying users.
- Fully or partially execute the user's program which can give an even more detailed explanation as the information being dealt with will be available.

Technologies in use: Haskell, Ruby, node.js, Angular.js, Java, git, mongoDB, redis