Eternal Programming

Roman Souzi
rnd@onego.ru Petrozavodsk, Russia

1. THE IDEA

Contemporary information technologies are moving very fast, leaving behind outdated software and hardware. It's already hard to find a 5.25 inch floppy drive to read the data stored on it ten years ago. And running programs written for old computers is nearly impossible.

How can an individual or an institution make sure their data-formats and data-processing techniques will survive centuries from now?

The approach proposed in this article assumes very little about future computers. Namely:

- Computers will have programming languages capable of manipulating texts and doing simple arithmetics;
- Data as a stream of bytes will be readable from physical media;
- There will be people who could read then ancient human language and could program computers;
- Byte will have no less than 8 bits;
- Latin letters and ten digits are still in use.

If these conditions are met, it is possible to use a proposed language (I call it Eternal). The core translator of Eternal must be easily implementable in nearly any computer language in less than a day from the strict specifications written in human language.

Then any program written in this language could be runnable. More than that, Eternal could evolve completely through libraries written for it. Those who want their processing techniques to be preserved for a long time will need to store these libraries together with their programs.

One feature of Eternal must be the invariance of it's specification. There must be only one version of the specification for the core interpreter. On the other hand, libraries can evolve, because every Eternal program will include all source code with it and thus will be runnable.

One of the concerns is preserving kinds of information other than text: sounds, graphical images, video, 3D environments. This could be addressed by additional specifications for the environment (I call it Perpetual), which will interact with Eternal and interpret multimedia data for the user.

Third component, code-named Immortal, will give artificial intelligence (AI) capabilities to Eternal and will be written purely in Eternal. However, Perpetual could be used to represent multimedia data for it as well. Immortal opens broad possibilities for preserving human expert knowledge.

The closest candidate for being Eternal is Forth, invented by Charles Moore [1]. The language has a simple core interpreter and can be evolved, even syntactically. However, Forth assumes too much about underlying hardware and probably is not what average programmer could make in a day.

The most useful applications of Eternal could be:

- file archivers and compressors
- cryptographic algorithms
- graphic, sound, video format converters
- hardware system emulators, especially CPU emulators
- AI knowledge-base and expert systems

Eternal specifications, reference implementations in different languages and libraries must be in Public Domain. As far as I know, there is nothing comparable to the Eternal idea.

2. ACKNOWLEDGMENTS

I thank Mr. Dennis E. Hamilton for fruitful discussion and review of this note on the Eternal.

3. REFERENCES

[1] C. Moore. Forth: A new way to program a mini-computer. *Astr. and Astrophys. Suppl.*, 5:497–511, 1974.