Symbolic Figures

By Francesc Hervada-Sala (mail: francesc at the-text.net)

Published Dec 17, 2015.

URL: http://the-text.net/symbolic-figures

Symbolic Figures: From the Clay Tablet to the Text Engine

Future of Text Symposium. Palo Alto, California, Dec 9, 2015

Today I am going to talk first about some ideas regarding text, then about the application of these ideas to computing, and last about my research projects.

While writing, science, and computing are big achievements of human history, they also show important limitations at the present time. Whatever area you are interested in, you can be overwhelmed by the amount of material to read and not be able to keep up with it. Science is fragmented into many disciplines, each of which has its own language, even its own culture, and communication between disciplines is difficult and insufficient. Also computer software comes in separate applications that do not work well together.

It is interesting that we find the same kind of problems appear everywhere. Why is this? Well, writing, science and computing are not completely different things, they share a common basis. Fundamentally, all of them work with symbols. When we speak, we use symbols. When we write, we produce symbolic structures that become fixed. When we do science, we build symbolic expressions and check them against reality. And computers are machines that can store and manipulate symbolic expressions.

The key point is that all symbolic expressions have the same kind of structure. I call «text» an articulated symbolic figure. And any articulated symbolic figure is a text. My thesis is that any text can be represented using a simple formula that can be expressed mathematically. This thesis also suggests a plan for action. If we build computer systems that manage text, then these systems will allow us to manage all writings, all sciences, and more as an integrated whole.

Although this plan can be applied to any field, my research is now concentrated in computing. What is the current software paradigm? Software comes in separate applications, each of which has its own functions, its own data, and stores the data in its own files. This has many disadvantages. Data and functionality are redundant. For example, you have a spell checker in your word processor application, and another one in your email application, if it happens to be from another vendor. Both spell checkers have the same function, but they perform differently, have different user interfaces, and use different dictionaries. You add a word into a custom dictionary in one application and you miss it in the other one. Functionality is not only redundant, but also very often missing. You cannot spell check your document names, just because the programmers have not prepared this particular case, although your system has several spell checkers installed. With the current paradigm, applications become big and complex, because they must provide all functionality that the user needs in a particular context. Applications are

expensive to develop and difficult to use. Additionally, applications cannot be combined and the user is confined to the functions that were anticipated by the developers.

I propose a new software paradigm. There should be a text layer underneath all applications. The operating system would have a text engine. A text engine is a software artifact that can represent text structures and manipulate them. That is, it can represent arbitrary symbols and set cross-references between them, and it can query and transform these structures. When an application wants to store some data, it stores the data through the text engine. And when the application needs to retrieve some data, it queries the text engine. There are no more files at all.

Imagine such a system. You would install a spell checker and it would be available to all your applications. You install an application to take notes and you are able to add notes to any sentence, to a document name, to a timestamp, to a menu option. You change the name of a document and all references update automatically. You would be able to choose the application that you want to use with your data. If you are writing a document, you can open a mind mapping tool that shows the document's outline, you update the mind map and the word processor updates automatically. At present, interfaces between applications must be programmed for each pair of applications. This is expensive and, as a result, many interfaces are missing. In the new paradigm, each application communicates only with the text engine, this way every application can collaborate with any other application. Applications are smaller and can be combined, this is better for the user, and better for the developer.

I have been working on two research projects to explore this idea. The first project was a program that I wrote some years ago with the programming language Perl. It is called the Universal-Text Interpreter, and it is a document generation system. To use it, you enter the information in some source files and write a script to generate output files in several formats. For example, you write some articles and then you generate a website and a PDF file from the same source.

This program is the precursor to the Text Engine, which is my current research project. The first version of the Text Engine will be used as a document generation system, but later on some applications will be built on the top of it. For example, support for virtual files. With a virtual file system you will use a conventional application to work on a conventional file, but the data will not be stored in the disk, instead it will be managed by the Text Engine. This way you can work on the same data using different applications. Imagine a word processor and a spreadsheet both having access to the same data. And when you make changes in one of them, they reflect automatically in the other one.

With these research projects I am beginning to explore the principle of using a layer of text to integrate different components. The Universal-Text Interpreter has proven that my text definition can represent any kind of information. The Text Enginge will extend text usage to application programming. And in the future, this approach can be scaled to the level of the operating system, and why not even to the level of the Internet. Then we will have big collections of data that are well integrated and operations with this data that are simple and can be combined and work well together. I see this as an extremely promising path to follow.

Thank you.

—-oOo—-