INTERFACE

FLORIAN CRAMER

It seems as if the term "interface" is borrowed from chemistry where it means "a surface forming a common boundary of two bodies, spaces, phases".[1] In computing, interfaces link software and hardware in relation to each other and to their human users. So they can be

(a) hardware that connects users to hardware; typically input/output devices such as keyboards, and feedback devices such as screens or loudspeakers

(b) hardware that connects hardware to hardware; network interconnection points and bus systems for example.

(c) software, or hardware-embedded logic, that connects hardware to software; the instruction set of a processor, for example, or device drivers.

(d) specifications and protocols between software and software; i.e. "application programming interfaces" (APIs)

(e) symbolic handles which, in conjunction with (a), makes software accessible to users; i.e. "user interfaces", often mistaken in media studies for "interface" as a whole.

In a software studies context, only the three (c), (d) and (e) are of relevance.

Regarding (c), any piece of software is an interface to hardware. Computer programs could be seen as tactical constraints of the total possible uses of hardware. They constrain, for example, the combination of a CPU, RAM, hard disk, mainboard, video card, mouse, keyboard and screen with its abundant possible system states to the function of a word processor, a calculator, a video editor etc.. In other words, they interface to the universal machine by behaving as a specialized machine, breaking the former down to a subset of itself. This operation is linguistic because it reformulates the totality of available machine

[1]According to Webster's Ninth Collegiate Dictionary which dates the term to 1882

instructions into a new control –> language. This language acts as an "abstraction layer". It is either a subset of the total available instructions when it is Turing incomplete, or a redressing of them with different symbolic handles when it is Turing complete.

"User interface" and "programming interface" have not always differed. They used to be identical in many operating systems up to 8-bit home computers in the 1980s that booted into a BASIC programming language prompt, or MIT's Lisp machines which had a Lips programming environment as their user interface. Character-based shells such as the DOS and Unix shells can be used both as programming and user interfaces. The same as true even for graphical user interfaces when they are scriptable. But even if they aren't, they still act, as a matter of fact, as specialized symbolic computer control languages. The distinction of a "user interface", an "API" and a computer control language is purely arbitrary. It's just a nomenclature that more complex interfaces to computer functions tend to be called "programming languages" and less complex ones "user interfaces". Since the usage interface to a computer program is always symbolic, and involves syntax and symbolic representations for operations, it always boils down to being a formal language. Everything that can be said about software interfaces is therefore redundant vis-à-vis the entry on –> language.