Article
Robert Cemper · Oct 15, 2017 3m read

@Indirection and eXECUTE - why ?

As I mentioned in an early post the roots of COS were laid in the late 60ies of the 20th century.

Close to that time, BLISS was designed at Carnegie-Mellon University (January 15, 1970)
https://en.wikipedia.org/wiki/BLISS
Similarly in 1972 C language was written at Bell Labs.
https://www.bell-labs.com/usr/dmr/www/chist.html

Both were designed to overcome the portability limits of code written in assembly language.
While BLISS was running on DEC machines and vanished with Digital Equipment Corporation
C language and later C++ expanded on almost any processor technology until today.

Both had the need to pass along data by references as the memory was very limited
and processors were incredible slow (compared to today).
So not data but just pointer to data (=memory addresses) were moved along.

In C it looks like this:

```
double a = 10;  /* a variable of type double */
double *p;    /* a pointer to a variable of type double */
p = &a;    /* initializing the pointer */
```

https://beginnersbook.com/2014/01/c-pointers/

The concept in BLISS is pretty much the same with slightly different writing.
[*.pointer instead of *pointer*]
Both strong typed languages gave a strict syntax to wrap a
common programming method of all assembly languages.

And though decades went by we still find these concepts in COS.
INDIRECTION is just another incarnation of the very same concept.
In COS you don't have to pass pointers for variables but instead the name of the Variable or Global.
It doesn't do much more than inserting the name handed over into some code to be executed.
See all 5 variants of indirection here:
http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GCOSoperators#GCOSoperatorsi__ndirection

When COS started to be compiled the "pass by reference" syntax with the dot was introduced (.pointer)
[*with best regards from BLISS* 

]

It is not a big jump in mind:
If you can get the name of a variable dynamically;
Why shouldn't you also get a piece of code dynamically?
Typically having a specific type of check code stored with data definitions.
[*In the light of compiled code this may seem somewhat strange, but*
for pure interpreted code it's just code from a different source.]

The command I refer to is eXECUTE with a bunch of extensions nowadays.
http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=...

http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=...

The probably oldest example of this type of programming is a command line editor
with VT layout that is still part of every distribution of Caché. **XECUTE ^%**

It's easy to understand that this style of programming is hard to read hard to understand
and hard to debug as the variables, references and even code are volatile in your data.
In most cases, there is just limited to no chance to separate data error from programming errors.
The classic separation of data and code just gets lost.

To escape from this conflict around Objects a set of specialized Class Methods and Instance Methods
has been introduced to bypass this need for Dynamically Accessing Objects.
http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GOBJspecialcos#GOBJspecialcos_dynaccess

BUT:
If you examine generated code (for Objects, SQL, XML, ...) you may be confronted with this
style of programming as it is quite comfortable to be used with code generators.

#Caché #InterSystems IRIS