POSITION PAPER

Alistair Cockburn
Application Development Consulting Practice

For brevity, I trust that that the reader will accept that distributed (including concurrent) computation can be reduced ultimately to signal transmission (usually to places unknown) and switching. The resulting computation can be represented as a directed, acyclical graph of events such as sending, receiving and switching. The graph is recursively decomposable, since an "event" is an abbreviation for a graph, and so the ordering is not an absolute order, but shows only precedence (A precedes C, but there may also be a B that comes between).

A similar reduction is possible on object relationships, where relationships are also objects (things), comprised of collections of named roles. A role is also a thing, a name (or place in an order) associated with a thing. A relationship ends up being diagrammable as a directed, acyclical graph. The similarity of this graph and the ones representing computation end up being significant.
Nothing is actually left of an object except its identity. 'What it consists of' is a separate thing, that changes over time.

The purpose of this exercise is not just to perform yet another formal reduction, but to end up with a represention of systems and behavior that (a) can be distributed (b) has built-in annotations to various readers, including not just people and run-time environments, but also to compilers, analysis tools and other "program-reading" systems. The system will have to capture the notion of readership, belief and caring within the formalism.

I am able, so far, to build from a set of primitives that include: identity, identical, belief, caring, precedes, consists of, multiplicity, collection, negation and 'changing to'. Note that 5 of these are relationships. Relationships are so complex that they do not make a good atomic element, so it is useful that 'relationship' can be build from a few, simpler relationships. Here is the start of the construction, showing how the primitives relate...

A thing *is* (has identity), but WHAT a thing is changes over time. A thing may consist of things, perhaps in an order that some other thing cares about. Things participate in relationships. Verifying those relationships requires computation, which requires a machine and time. The thing that does the verification is at that point in a position to state that it believes the relationship to hold.

Time is important only in that a thing changes what it is. Ultimately, everything is a description, except for the changing. Although changing can be described, the changing itself is not a description, it is actual and personal.

Two apparently different things may be identical (share the same identity). It is not necessary to know that for it to be true. Since it takes time to verify that they are identical, and since 'what they are' changes over time, identification can be subtle (e.g., "Are you the same Tom who put a pin under Miss Jones in 5th grade?").

Sometimes we do not care about the identity of a thing, sometimes we care but do not know. The notion of a free variable involves us caring but not knowing. The notion of a collection being unordered requires not caring, as does statements like, "for any finite number of elements".

A shortcoming of most systems today is that they do not allow us to address different readers ("things which care") within the system. Smalltalk provides a language for describing execution. We need to be able to add commentary about <the method parameters, for example>. That commentary should not be intermingled with the execution description using syntactical conventions, as is done in C and C++, but should be a separate description, using identification ("being identical to") to attach to the desired items within the execution description.

Another shortcoming is that systems today mandate one and only one form of description. E.g., an object is equivalent to a finite state machine, but we are usually not permitted to describe it using FSM notation, we have to describe it using other notation. In a growing, distributed world, the language of description should be aimed at "readers", allowing multiple languages of description to coexist. Distribution and concurrency require that descriptions of computations and associated annotations be distributed around the network.

A third shortcoming is that most formalisms do not make the notions of "description", "machine" and "transmission" explicit, even though it is well established that what good programmers do is invent machines whose programming is simple in an intended domain. LISP is a language of descriptions, which relies upon machines for its computation, but never says what a machine is. Smalltalk is a language of machines, but is not explicit about transmission or about descriptions. The formalism here is aimed at capturing all three.

The idea of the above formalism is that computation and relationships are both descriptions and can both be described using a language of directed, acyclical graphs, with the added notions of caring, knowing, and so forth. The notion, "identical", allows us to separate fragments of the description for convenience, and to discover identical items over time. A machine is made from multiple, communicating parts agreeing to certain conventions among themselves.

...It is very difficult to describe these ideas in linear text. They beg for drawings, since they work with graphs... I shall just finish with a couple of fragments that can be described by characters and character graphics. Pardon the primitive rendition.

Fragment example 1:

Thing A: ...a parameterized description of behavior...
Thing B: "A cares that its second argument is an integer"
Thing C: "A should only be used by subclasses".

Fragment example 2, showing identical over a distance:
(connecting lines are "identical")
(a "name" is a thing, shown in "..")
(a role is a pair: name, thing)
(here, I am dad, you are son, the two roles are part of a father/son relationship, and part of the family relationship. Each of these is actually so, although in a distributed system, it may some time to discover that.)

```
                        "father/son"       "family"
         _______     _____________    _________________________________
        | "dad" |    |             |  |                                 |
 me---------*   |-----------*-----------("father" *)                    |
        |_______|    |             |  | "mother" *                      |
                     |             |  |            ____________________ |
         _______     |             |  | "children"|                    ||
        | "son" |    |             |  |           |* * * (some number) ||
 you--------*   |-----------*      |  |           |____________________||
        |_______|    |_____________|  |_________________________________|
```

Alistair Cockburn   Application Development Consulting Practice
E-mail:ALISTAIR at RHQVM21 alistair@vnet.ibm.com USIB2QFF at IBMMAIL
Postal: IBM, m/d 248, 150 Kettletown Road, Southbury, CT 06488
Phone(203)262-2319 Tie-line:376 Fax:day=(203)262-5957 night=(203)262-5955