It's Alive!

Animated Programming With Cells

Industrial Strength

Applied over eight years to a wide variety of applications

- Educational software
- Double-acrostic crossword puzzle
- Physical simulation of a pendulum
- Clinical drug trial management
- RoboCup Soccer Simulation League

Fast

As much effort has gone into performance as power

- Propagation stops when any calculated dependency recomputes the same value
- Dependencies exist only for other cells actually (not lexically) consulted
- No dependencies are recorded for cells declared constant (implicitly, by being initialized to anything other than a cell)
- Ruled cells which happen not to consult another cell are optimized away

Faster

- Synapses can minimize recomputation
- eg. (> (^pressure (sensor self) (fSensitivity 10)
- Lazy evaluation
- Intelligent propagation to the logically minimum number of state points
- Cells for DEFSTRUCT on the way

Correct

Directly addresses program correctness

- Eliminates the entire class of bug in which internal program state is inconsistent
- Guarantees all state affected by any program input will be recomputed...
- ...in the right order.

Easy

The Prime Directive: Productivity

- Dependencies are identified automatically
- ...and reliably, and dynamically.
- Propagation is handled automatically
- Discipline keeps users from breaking the paradigm, inadvertently or otherwise.
- Declarations are deterministic, hence comprehensible and predictable.

Expressive

No limit on the semantics associated with a cell.

- Model population is itself mediated by cells
- Instances arranged in a navigable workspace have access to any slot of any other instance
- “on change” observer call-backs are generic functions dispatched on the instance, the slot- name, new value and old value

More Expressive

Varieties of underlying cell slot

- “Ephemeral” cell slots revert to an identity value after propagating to support event-like data
- Delta cell slots are second-order, propagating if a computed value is not the identity
- For normal cell slots, default “unchanged” test of EQL can be overridden

More and More Expressive

Varieties of cells for same cell slot

- Drifters begin with a value and can be SETF'ed. Rules are taken as a delta to be added (with “addition” user-definable).
- Cyclicals allow limited amount of two-way dependency
- Aggregates support composite structures in a single slot value

The Database: It's Alive!

Cells and a persistent CLOS database, happy together in a GUI workgroup app maintaining a document database.

- Cells of view instances watched persistent slots of DB instances
- Dynamic slots of persistent instances reflected the state of persistent slots of instances
- Cells of view instances watched the population of persistent tables
- Persistent slots watched other persistent slots and the population of persistent tables

More, More, and More Expressive

User-definable Synapses mediate change propagation

- Two rules for each synapse
- For efficiency: Do I propagate?
- For expressiveness: What should I propagate?
- Example: delta synapses return the change in the slot mediated, and take an optional sensitivity parameter to reduce propagation

Plays Well With Other (Code)

Seamlessly integrated with the HLL

- Works within HLL object model
- Rules written in arbitrary HLL
- On-change callbacks written in arbitrary HLL
- (I'd say “Lisp” and “CLOS” but Cells has been ported to Java, C++, and Python)

Makes OO Work

Finally objects deliver on the promised Grail of re-use

- Functions do for slots what they did for function arguments: higher order parameterization
- Different instances of the same class have wildly different rules
- No need to be forever defining new classes (or to give up on OO and write huge methods driven by literal values in slots

Self-documenting

Makes explicit the dataflow implicit in inanimate paradigms

- Declarative model (and discipline) forces full semantics behind a slot to be expressed in one place, in one rule.
- Diagnostics could analyze run-time dependencies which actually arise

The Lisp of Paradigms

Been around for a long time, used and loved by a happy few.

- SketchPad, 1963
- ThingLab
- VisiCalc
- Steele's 1984 thesis on constraint programming
- The entire constraint logic programming domain
- Access-oriented programming
- Kaleidoscope, Garnet's KR, COSI, Cells
- OpenLaszlo
- Flapjax, functional reactive programming

So What Went Wrong?

"Given that constraints were used as early as 1962, why were not these ideas explored further, rather than waiting ten to fifteen years?"

Guy Steele, on prior art to his new constraint language.

Twenty-five years ago.

A Bridge Too Far

Over-enthusiasm for an exciting technology

- Partial constraints: “less than 100"
- Multi-way constraints
- Given: (eql a (+ b c)), now change “a”

Stop. Wrong way. Go back.

"One common problem for users of the CLP systems in use is the sometimes unpredictable behavior of the constraint model: even small changes in a program or the data can lead to a dramatic change in the performance. This is because the process of performance debugging, designing and improving constraint programs for a stable execution over a variety of input data, is currently not well understood. Related to this problem is the long learning curve that novices have experienced. While simple applications can be built almost immediately, it can take a long time to become familiar with the full power of a constraint system."

Rossi, Francesca. "Constraint Logic Programming"
University of Padova, 1997.