Desirerata

Technology Platform

PHYSICAL INTERFACE

Detect position of lasers	✔ will do
Illuminate walls and objects	✔ will do
Track location of objects	✔ will do!
High-resolution illumination	✔ will do
Scan walls and objects	~ in a limited way (not complete coverage)
Detect and parse voice	~ probably in a limited way, for voice commands
Print or fabricate objects	~ will do, but probably not in a fully dynamic way
Actuate and move objects	~ in limited domains (xy table, microrobots)
Detect laser identity / buttons	✖ probably not for now
Track people's location, gestures	✖ probably not for now
Track manipulation of objects	✖ probably not beyond position/orientation
Morph objects	✖ not yet

METAPHORS

Attachments, instead of virtual filesystems and databases. Computational processes and data collections are virtually attached to physical objects.

I'm less attached to the word "attachments", but it's very much the case that every process and state variable is associated with a physical object.

Responding to the environment, instead of messaging. Processes are coordinated not by direct communication, but by influencing the physical state (e.g. moving) or virtual state (e.g. adding data to a collection) and observing such changes around them.

Observables and observation are front-and-center, and should be the primary form of influence. But we're also adding messaging for the times when you need to push.

Seeing the world, instead of querying a database. Objects look around themselves in space and time, and notice changes of interest. Objects see each other.

Observables represent this concept.

Dynamic ether? Perhaps the space between objects (the air, the background or "game board") can run processes and hold data.

Not as a primitive, but a recognizer can recognize a volume of air, or a game board, as an "object", which can then run processes and hold data.

WORLD MODEL

Literally global. Every object in the world can be can be referenced with a unique id.

Probably not for now. Tagged objects will be unique within our room, and untagged objects do not have a persistent virtual identity.

Gracefully incomplete. The real world is truth. The computer's model of the world is necessarily incomplete, and perhaps even inferred probabilistically.

Recognizers and observables are in this spirit, or can be. For now, I don't expect much probabilistic inference at the system level (just contained within limited domains such as vision and voice processing).

Query across space. Objects can see other objects. Queries can involve spatial scope and orientation.

Yes, via location observables.

Query across time. Objects can see everything that has ever been. Queries can involve temporal scope, or can operate over time (like signal processing filters).

Would like to aim for this. Observables-as-streams is a first step, observables-as-of-a-point-in-time is a next step.

Query across possibility. Objects can fantasize. Queries can involve simulated future scenarios in parallel worlds.

Probably not for now.

Provenance and influence. Where did this data come from, and where did it go? It should be possible to reconstruct an entire chain of events.

Probably in the form of: an observable value is tagged with the observable values and/or message that it is a "reaction" to, as well as the process that produced it. This tag is visible from the meta-system. This will probably require the participation of the interpreter, instead of happening at system-level. (All this data is currently collected in v2, but not made visible. How to make it visible is its own project.)

First-class people? Perhaps a person should not be an "object".

Don't know yet how or whether to model people in the system.

INTERPRETER / RUNTIME

Now thinking about this as a multilingual system, with multiple interpreters that interface with a common object model.

Simple implementation, in the STEPS sense. Can be looked at and understood. Perhaps realized physically.

Interpreters are objects, so they can be as spatial and physical as any object. An interpreter can be a poster, whiteboard, exhibit, or whatever.

Flexible. Instead of provisions for anticipated features, the language should allow features to be added as their need is recognized.

Flexibility comes from allowing for arbitrary languages on top of a flexible object model.

Instant update. One can make fluid code changes (e.g. dragging a slider) and the process should update immediately and fluidly.

Programs are observables, and observables are streams. Interpreters will observe changes to the program, and can update appropriately.

Full control. We can change the runtime into what we want.

Some interpreters, such as Node and Python, we'll be stuck with. As we gravitate toward our own interpreters, we'll get more control.

Inspectable. We can see and visualize the inner workings of the interpreter.

Likewise, we'll start out with opaque interpreters, but as we gradually make our own within the system, they'll be as inspectable as anything else in the system.

FFI? We need some way of incorporating foreign code when necessary.

At a coarse-grained level, behaviors can be in any language and can communicate with behaviors in other languages. At a fine-grained level (e.g. calling a simple javascript function from a custom language), not sure yet.

LANGUAGE

Most of these things are up to the individual languages, but there shouldn't be anything about the object model that prevents them.

First-person objects. The author thinks from the perspective of the object.

First-class space, time, and possibility. The author thinks in terms of what the object sees, has seen, and could see.

Evocative queries. Query syntax is not "blind", but is seen in the context of what is being queried.

Richer than text. Images, graphs, etc., can be part of the code. (And not merely as comments.)

Directly-manipulable. The language is designed for making continuous changes (e.g. dragging a slider.)

Direct referencing of physical objects. Programming involves pointing at relevant objects and data in the world, not typing their names

Transition path away from screen. Towards programming by manipulating physical objects.

Interpreters with arbitrary programs are intended to support this path. Programs can be any amount of textual or physical. But this will require a good toolkit for making interepreters.