Externalizing Co-occurence: Potentializing and Reifying Interactions

Antranig Basman
Raising the Floor - International
amb26@ponder.org.uk

Philip Tchernavskij
Université Paris-Sud
ptcher@lri.fr

Simon Bates
OCAD University
sbates@ocadu.ca

ABSTRACT

We present a new taxonomy for describing the conditions and implementation of interactions, based around the notion of co-occurrence and entanglements. We explain that in order to support open ecologies of function and fabrication, we need new conditions, quantities and goals in computational systems. To support ownership and inhabitability of digital artefacts, we need to support externalised designs and design tools. This goes against traditional designs which stress the importance of information isolation and specialisation in the roles of system components and designers. There are many ways of conceiving this new design space, and we present three dominant metaphors which we have employed so far, based on chemical reactions, quantum physics and cooking. We exhibit different systems which we have implemented based on these metaphors, and sketch how future systems will further empower citizens to design and inhabit their own interactions, express ownership over them and share them with communities of interest.

ACM Reference format:
Antranig Basman, Philip Tchernavskij, and Simon Bates. 2016. Externalizing Co-occurence: Potentializing and Reifying Interactions. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 6 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 EMBODYING INTERACTION

To support human activities, what computers do must be about more than just computation. In a reframing of computational activity, Wegner [11] argues that "Interaction is more powerful than Algorithms". We support Wegner's problematisation of computation, that "Algorithms and Turing Machines, like Cartesian thinkers, shut out the world during the process of problem solving". Taking computers and computation beyond the role of "problem solvers", we want to embed their activities in the real world as part of an *open ecology of function* [7]. We want to facilitate and empower creative networks to curate and share artefacts of interest. To reach this aim, we need to spend more time teasing apart the nature of what interaction **is**, what it relies upon, and how its prerequisites can be organised and filtered in order to allow interactions themselves to take the role of first-class artefacts of interest. This is the goal of this paper.

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

1.1 Ecologies of function and fabrication

Basman [3] and Clark and Basman [7] introduce the notion that artefacts could and should take part in open ecologies. The ecology of **use** or **function** reflects how an artefact comes to be recognised and used amongst a collection of similar artefacts with similar functions — for example, the ecology of spoons presents obvious and related affordances based on their relative size, shape and materials. The ecology of **construction** or **fabrication** establishes relationships amongst creators which may or may not mirror those amongst the artefacts which are being constructed — different craftsmen may specialise in spoons of different materials, different dimensions or context of use. Expert users may have the option to make connections to relevant members of the fabrication ecology and perhaps start taking on some of their skills. However, what is good for spoons is not good for software — even the most basic of the skills necessary to construct acceptable software will remain out of reach of the vast majority of citizenry who could benefit from their use.

1.2 Open Authorship

So-called end-user development has been a goal of many communities over decades [10], but we argue that an important impediment to useful results has been a failure to concentrate on the real materials [3] and function of software development, both from the point of view of these ecologies but also from the resulting refined taxonomy of activity that they will reveal within the everyday expressions of software engineers. A clearer statement of this ecological goal is the "open authorial principle" described in [4] — that any expression by any one author of a system can have its effect replaced via an *addition* to the design by another author.

1.3 Interaction and co-occurrence

In this paper we will focus our gaze on what has been called "interaction" — encompassing not only what has traditionally been considered the interaction between the computer and the human, but also to the wider modelling of interactions in the world that the use of computer-mediated artefacts will involve.

Before an interaction can begin, there is a prerequisite condition that we will name **co-occurrence**. Certain elements of interest must have been assembled in a "suitable proximity" — this proximity may be physical, informational or take some other form which makes them conveniently available to each other or the user.

Here are some examples of interactions, whose start is triggered by a suitably defined co-occurrence:

- Two sensors are plugged into the same machine
- A finger is touching a screen
- A colour-picking instrument is targeting a particular pixel

- A user's gaze is being tracked by a camera determining its intersection with an element shown on screen
- A group of musical instruments is assembled in a sufficiently small area that they may mutually communicate wirelessly
- A user whose wheelchair carries a device which facilitates communication is brought near to a kiosk running an application they wish to interact with

In each case, the conditions for co-occurrence being satisfied signal that an interaction may potentially begin. In any realistically-sized system, such co-occurrences may be detected with high frequency or even continuously. This requires a further round of interaction between the user and the system which detects and transmits the co-occurrences, which we might name the *co-occurrence engine* or *entangler* (see section 1.6 for variant terminology). The user could select from a menu of available co-occurrences, perhaps with suggestions emphasising choices that they or members of their community of interest have frequently made in the past. In simpler, or classical interactions initiated by means of a dedicated pointing device, it would be appropriate to make the classical choice of initiating any available interactions immediately.

1.4 Signalisation of co-occurrence

It helps greatly in constructing externalisable systems if their constituents take an *integral form* — in the language of functional reactive programming, we would say that they form a *signal*, which has a (in our case complex, structured) value which is in theory observable at any instant in time. We can then cast the work of the entanglement instantiator as a pure function of the co-occurrence signal, which maps detected co-occurrences into entanglements for as long as they last, and subsequently maps them back into non-existence[1]. This signal can then be conveniently transported around different sites of a distributed system, allowing the instantiation machinery itself to be implemented wherever it is convenient — perhaps at sites where computation is economically available, or perhaps where communication with a crucial system element or the user has particularly low latency.

1.5 Anatomy of an interaction: Co-Occurrence and Entanglement

After a suitable co-occurrence has been recognised and selected for activation by entering it into the co-occurrence signal, it must be acted upon to initiate an interaction. At this point there are various sets of terminology we will use to describe the process, governed by different sets of metaphors. These are elaborated in section 1.6, but for the current discussion we will refer to the entity created in order to represent the interaction process as an **entanglement**. The entanglement has a lifetime coextensive with the interaction, and represents it as an addressable element of the system's runtime. Note that whilst the beginning of the lifetimes of the co-occurrence and entanglement will invariably agree, there may be interaction models in which the the entanglement persists beyond the co-occurrence and follows the agents as they diverge, perhaps to be dismissed as a result of some special signal issued by any of the participants.

There are some conditions on entanglements which result from our open authorship goals described in sections 1.1 and 1.2.

Firstly, entanglements themselves must be first-class elements of the system in their own right — and capable of giving rise to further co-occurrences and hence entanglements. If, for example, the underlying paradigm of the design were object-orientation, this would imply that there is an "entanglement object" for the lifetime of the interaction which it represents.

Separating the description of interaction into co-occurrence and entanglement is necessary in an open ecology of function. Shifting responsibility away from the programmer, the purpose of this separation is to ensure that it is no longer a programmer's job to fully specify all the entities that will be available in an "interactive system" throughout its lifetime.

When communities bring together various kinds of sub-systems, this separation of description is necessary to allow the integration of these systems — since this integration creates the possibility for encounters between groups of elements which had not been part of the design of either system. It may be the task of a third community to describe and orchestrate such interactions, and each system must be designed in such a way that this third-party integration is possible. This requires, in addition to the anatomisation of the interaction process that we describe here, suitable *public coordinates* for each element of both systems that allow the expression of the required co-occurrence rules. Following Kell [9], we propose that user software should form an *integration domain* where useful interactive artefacts can be coupled to construct personal and communal interactive systems.

The origin of the notion of interaction as entanglement comes from the consideration of the philosophical idea that a tool is not so much a thing in itself as a thing that an agent has brought into a tool-like relation. If we cannot know at the time software is produced all entities that will exist in a given software system, we cannot know what interactions will exist either. Interactions are by their nature boundary-crossing: The ability to write is not contained within a pencil, writing is a thing that a person does by tracing a pencil over paper. This implies that a software substrate for an open ecology of function can be told to opportunistically recognise the co-occurrence of elements with particular properties, and instantiate behaviour that couples the state of these elements for a time.

1.6 Variant metaphors for mapping the entanglement process

Since this is freshly mapped conceptual territory, some diverse metaphorical structures have been applied to describe the elements operating the entanglement process. The base term "co-occurrence" is fairly generic and not tightly bound to any particular mapping, but other metaphors for viewing the nature of the process lead to different names for its elements. These are described in the following subsections and illustrated in table 1

1.6.1 Chemical metaphor. In this metaphorical structure, we describe the prerequisites for the interaction as *reactants* and the

[1] However, not all interactions may naturally be integrated in this way. Some entanglements may be terminated by an arbitrary signal issued by any participant — systems which support this facility would have increased costs of communication and reduced transparency.

resulting structure representing the live reaction as the *product*. This metaphor is helpful in aligning us with the SMIRKS reaction transformation language [1], developed in order to describe not only the molecules which are required for a particular chemical reaction to proceed, but also the exact relationship between atoms in the product and those in the reactants[2]. This is helpful as a generalisation of the regular-expression-like SMARTS language for encoding predicates on molecules, as well as showing an example of how declarative and publicly intelligible notations for reactions can be written and operated by a running system. Areas where this metaphor breaks down for us include the fact that chemical reactants are typically consumed during reactions, and the products are fabricated from their components — whereas for the kinds of reactions we are centrally interested in, the product typically mounts *references* to the reactants which are typically unchanged by the process of constructing the product itself (although they may be modified by interactions which occur during the entanglement).

1.6.2 Quantum metaphor. In a metaphor taken from quantum physics, we describe the participants in the reaction complex as being *entangled*, and speak of the resulting complex as an *entanglement*. The machinery which instantiates and transmits descriptions of these complexes is then named an *entanglement engine* or *entangler*. This metaphor is helpful in situating the purpose of the complex as being primarily informational, and also one which is used to mediate communications and relationships as long as the interaction persists — as well as holding the possibility that once the participants retire beyond their interactional horizon (they cease to co-occur), that the entanglement complex is torn down. In the quantum theories which feature them, these entanglement complexes are considered to mediate all forces by which macroscopic bodies interact with each other. This metaphor better maps the "reversible construction" aspect of the entanglement complex than the chemical metaphor, but provides fewer implementation clues.

1.6.3 Cookery metaphor. In this metaphor, the available elements are considered as ingredients in a recipe. The co-occurrence of particular groups of ingredients may make the cooking of different dishes available. This metaphor is more useful at the level of the user of an overall system, who may well be familiar with the process of rummaging through their kitchen cupboards in order to determine what ingredients have co-occurred there. It may be that the user habitually wishes to cook particular dishes when faced with a particular co-occurrence, or instead wishes the system to surprise them through suggesting a previously unvisited combination. This metaphor suffers from the same deficiency as the chemical one, in that the ingredients are considered to be irreversibly consumed through the reaction, which is an unsuitable disposition in an open informational ecology.

2 EXAMPLES OF ENTANGLEMENT AND CO-OCCURRENCE ENGINES

We have built several examples of engines operating interactions, which highlight different aspects of our ultimately desirable idiom with varying degrees of fidelity.

[2] An early precedent for the idea of using chemical reactions as a computational model appeared in [6]

2.1 The Co-Occurrence Engine

This is an implementation aimed at facilitating the use of multiple physical devices and sensors coordinated at or near the same device. As such it currently operates a relatively coarse-grained co-occurrence criterion based on the simultaneous presence of elements advertising particular capabilities, encoded as namespaced strings (in Infusion, *grade names*).

The goal of the Co-Occurrence Engine is to enable dynamic configuration within an Infusion [8] system, based on the presence or absence of Infusion components. We have used it with the GPII Nexus to facilitate exploration of designs for an inclusive science lab (see section 3).

The Co-Occurrence Engine monitors a collection of Infusion components and is configured with a set of recipes. Each recipe contains two sections: a set of reactants and instructions for constructing a product. Each reactant has a name and a match rule. The reactant name may be used within the product to refer to the matched component. In the current implementation, a single match rule is available: match components based on namespaced strings present on the components (grade names). An example recipe from the Nexus Science Lab is shown in Listing 2.

The Co-Occurrence Engine constructs and maintains product components based on the co-occurrence of reactants. If components are added that bring into being new co-occurrence situations with matching recipes, the corresponding products are constructed. And if components are destroyed that remove previously present co-occurrences, the affected products are also destroyed.

2.2 Entanglement Engine

This implementation is intended to demonstrate as a general-purpose substrate for user software. The Entanglement Engine mimics the role of the CSS engine in a web browser, transposed from managing the presentation of content to managing its behaviours.

Rather than operate on web documents, the Entanglement Engine is implemented to operate on a structured data world inspired by the DOM, extended to also represent browser-external elements, such as mouse cursors and input devices. The engine is configured with *entanglers*, which are largely similar to Co-Occurrence Engine recipes, with a few notable exceptions:

Firstly, if an entangler is triggered, the engine produces an entanglement, a relationship coupling the state of the entangled elements, which are called components. These relationships currently take the form of programming abstractions appropriate to the concrete interaction, e.g., event listeners, reactive functions, state machines a la [2], etc.

Secondly, entanglers include a list of predicate functions that must hold before an entanglement is created. These predicates are used to describe different kinds of co-occurrence. For example, one entangler may require that its components geometrically overlap, while another may require that one component contains a reference to another. The set of available co-occurence signals is currently small, but our intent is that entanglers can be configured to trigger based on the signals appropriate to various interaction paradigms, e.g., device proximity for cross-device systems, or gesture detection for gestural interaction.

Description	Chemical metaphor	Quantum metaphor	Cookery metaphor
The characterisation and detection of those elements which might participate in an interaction		Co-occurrence	
The description of the participating elements, the process which they enter into, and the identity of the result		Entangler	Recipe
Machinery which operates the process	Reactor (Co-occurrence engine)	Entanglement engine	
The elements which potentially or actually participate	Reagents	Entanglement components	Ingredients
The process of combining the elements	Reaction	Entanglement	Cooking
The complex resulting from combination	Products	Entanglement	(Dish)

Table 1: Terminology arising from variant metaphors to describe the entanglement process

```
{
    "type": "gpii.nexus.recipe",
    "reactants": {
        "phSensor": {
            "match": {
                "type": "gradeMatcher",
                "gradeName": "gpii.nexus.atlasScientificDriver.phSensor"
            }
        },
        "collector": {
            "match": {
                "type": "gradeMatcher",
                "gradeName": "gpii.nexus.scienceLab.collector"
            }
        }
    },
    "product": {
        "path": "sendPhSensor",
        "options": {
            "type": "gpii.nexus.scienceLab.sendPhSensor"
        }
    }
}
```

Listing 2: A Co-Occurrence Engine recipe from the Nexus Science Lab

2.3 The Embranglement Engine

This is a proof-of concept implementation (using the term "embranglement" in place of "entanglement") which operates in a toy world of three agents in a web browser. It was designed to clearly exhibit the signalised form of the co-occurrence condition, which is displayed as a JSON document in the middle pane of the UI, updated at each frame of the interaction. Figure 1 shows a screenshot where the three agents have been deemed to co-occur and an embranglement has been instantiated coupling them together. In this case the co-occurrence condition is geared through "focus/nimbus" relations. [5] describes a model of interaction where an agent has multiple zones of perception — the "focus" of an agent is the region of its own perception, and its "nimbus" is the region from which it can be perceived.

This illustration shows three point agents, coloured red, green and blue which each have a focus and nimbus which is circular and concentric, with the focus nested within the nimbus. The agents enter a 3-way entanglement when the focus of each intersects the nimbus of each pairwise. The construction state of the entanglement is illustrated by a white triangle joining the agents, with the unique identifier of this particular entanglement instance annotating it. This implementation was made in the Infusion system [8] developed by the Fluid project, a configuration dialect which makes substantial use of JSON structures and the natural alignment and coordinate system they induce in order to support the aims of open authorship.

```
/* An entangler for wiring up a mouse and something with a position to move as the mouse does. */
export const mousemove = {
    name: 'mousemove',
    components: {
        mouse : ':mouse',
        moved : '[position:2d-coordinate]'
    },
    configuration: ['atLeastOneComponentTag'],
    on_start: 'reaction getHash -> moveCursor',
    actions: {
        getHash: function() {
            return hashCode(this.find('mouse'));
        },
        moveCursor: function(hash) {
            /* function body omitted */
        }
    },
}

/* Spawn a cursor for each mouse that connects to the system. */

export const makecursor = {
    name: 'makecursor',
    components: {
        mouse: ':mouse'
    },
    configuration: [[['mouse'], 'hasNoCursor']],
    on_start: 'makeCursor',
    actions: {
        isInEntanglementFromEntangler: function(thing: Data, entangler: string) {
            /* function body omitted */
        },
        hasNoCursor: function(components) {
            /* function body omitted */
        },
        makeCursor: function(components) {
            /* create a cursor */
            let cursor =
                d(':circle.mousemove-moved.shapefocus-focus.dragging-leader', {
                fill       : d(':color', 'transparent'),
                stroke     : d(':color', '#000000'),
                strokewidth: d(':number', 2),
                radius     : d(':number', 8),
                position   : d(':2d-coordinate', {x: 0, y: 0}),
                targets    : d(':id-list', [])
            });
            /* add it to the document */
            let cursorMountPoint = findFromNode(components.mouse, ':root', ':renderables')[0];
            cursorMountPoint.get('content').push(cursor);
            /* entangle the cursor and the mouse using the mousemove entangler */
            this.entangle({
                mouse: components.mouse,
                moved: cursor
            }, mousemove);
        }
    }
}
```

Listing 3: Two Entanglement Engine entanglers

As a result, the entanglement is indeed on an even footing with the original agents, and meets the criteria in section 1.5 for being the basis of further co-occurrences, as well as being signalised in a

form which could easily be advertised and manipulated outside the browser process which hosts it.

In this model of interaction, we can simplify the transmission of the state of co-occurrence and entanglement, since this state is simply a function of the position an orientation of all the agents. This state can then be easily transmitted through a distributed system in the form of the JSON document ("embranglement signal") shown in the right pane. As we note in section 1.4, not all interactions may be signalised straightforwardly in this way, although an Infusion component tree as a whole enjoys a natural externalisation which could still be relied on in the case of interactions which lacked a natural integral delimitation.

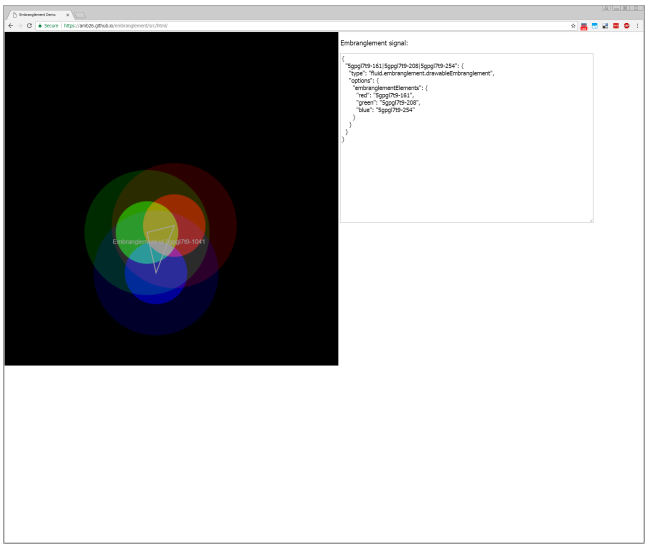

Figure 1: Screenshot of in-browser embranglement engine at the point where an embranglement has been instantiated

3 CO-OCCURRENCE IN THE WORLD OF THE USER

Here we will describe how one of our engines, the co-occurrence engine, has been employed to construct a system and interaction meaningful to end users in a physical context, the Nexus Science Demo.

The goal of the Nexus Science Demo is to explore designs to make science labs more inclusive. We use the GPII Nexus [7] to connect lab sensors to a range of different presentations and interactions. This helps a student pick the interaction that works best for them. For some students this might be a visualisation and for others it might be a sonification, or a combination of different presentations.

Drivers for three sensors were developed: a USB pH sensor, a USB electrical conductivity sensor, and a Raspberry Pi based temperature sensor. And six presentations were developed: a numeric dashboard showing the values of all connected sensors, a bar-graph visualisation, a coloured pH visualisation annotated with the values of common substances, a coloured temperature visualisation, a general ranged-value sonification, and a pH sonification.

The sensor drivers work by maintaining peers within the Nexus. When a sensor becomes available (for example by connected a USB based sensor to a computer running a suitable driver, or by adding a Raspberry Pi to the network), a peer is constructed within the Nexus, and a WebSocket connection is established to stream sensor value updates. When a sensor becomes unavailable, the peer is destroyed. In this way, the availability of each sensor is indicated by the presence or absence of a peer within the Nexus.

The Co-Occurrence Engine is used to decouple the sensor drivers from the presentations that students use to interact with the data. The sensor drivers need only make their data available and the Co-Occurrence Engine recipes organise the data for the presentations. Presentations expect a central data store with an entry for each sensor, together with metadata such as name and value range. The Co-Occurrence Engine is configured with one recipe per sensor type, each with 2 reactants: the sensor that the recipe is for, and the presentation collection data store. When these reactants are present, the recipe constructs a product that relays the sensor data to the collection data store, in a format expected by the presentations.

The Co-Occurrence Engine facilitates dynamic connection and disconnection of sensors, as student needs change. When a new sensor is connected, the driver will construct a peer for the sensor. The Co-Occurrence Engine will then construct any product components needed to make the sensor data available for presentation. Once the sensor data is available, the student may then select from the available data presentations. When they are finished, they may disconnect the sensor. The driver will destroy the peer and the Co-Occurrence Engine will tear down the products that it had previously set up.

4 CONCLUSION

We have introduced a new taxonomy for the constituents of interaction, and set new goals that must be met by systems offering it. We have exhibited variant metaphors and variant concrete implementations for such systems, contrasting the relevance of each for different tasks and contexts. We have shown how such reconception of the process of interaction can lead to more externalised designs, and a more open set of choices for who can make decisions about how a system should be, and how it is mapped onto the world.

REFERENCES

[1] SMIRKS - A Reaction Transform Language. (????).
[2] Caroline Appert and Michel Beaudouin-Lafon. 2006. SwingStates: Adding State Machines to the Swing Toolkit. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06)*, Jeff Pierce (Ed.). ACM, New York, NY, USA, Article 1. https://doi.org/10.1145/1166253.2180954
[3] Antranig Basman. 2016. Building Software is Not a Craft. In *Proceedings of the Psychology of Programming Interest Group*.
[4] A. Basman, C. Lewis, and C. Clark. 2018. The Open Authorial Principle: Supporting Networks of Authors in Creating Externalizable Designs. In *Submitted to Onward '18*. https://github.com/amb26/papers/blob/master/onward-2016/onward-2016.pd
[5] Steve Benford and Lennart Fahlén. 1993. A Spatial Model of Interaction in Large Virtual Environments. In *Proceedings of the Third Conference on European Conference on Computer-Supported Cooperative Work (ECSCW'93)*. Kluwer Academic Publishers, Norwell, MA, USA, 109–124. http://dl.acm.org/citation.cfm?id=1241934.1241942
[6] Gérard Berry and Gérard Boudol. 1992. The chemical abstract machine. *Theoretical computer science* 96, 1 (1992), 217–248.
[7] Colin Clark and Antranig Basman. 2017. Tracing a Paradigm for Externalization: Avatars and the GPII Nexus. In *Companion to the First International Conference*

on the Art, Science and Engineering of Programming (Programming '17). ACM, New York, NY, USA, Article 31, 5 pages. https://doi.org/10.1145/3079368.3079410

[8] Fluid. 2018. Fluid Infusion Documentation. (2018). http://docs.fluidproject.org/infusion/development/

[9] Stephen Kell. 2009. The Mythical Matched Modules: Overcoming the Tyranny of Inflexible Software Construction. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA '09)*. ACM, New York, NY, USA, 881–888. https://doi.org/10.1145/1639950.1640051

[10] Fabio Paternò and Volker Wulf. 2017. *New Perspectives in End-User Development*. Springer.

[11] Peter Wegner. 1997. Why interaction is more powerful than algorithms. *Commun. ACM* 40, 5 (1997), 80–91. https://doi.org/10.1145/253769.253801