Co-Occurrences and Entanglements: A Programming Model for Open-Ended Interactive Software

ANONYMOUS AUTHOR(S)

Tools for creating highly adaptable software, such as live programming environments, enable the ongoing adaptation of an interactive systems to fit diverse and changing use situations. However, state-of-the-art adaptable environments provide limited support for adaptations that affect how objects interact with each other. By contrast to other types of system refinement, these kinds of adaptations tend to be more rigid and brittle, because they create strong couplings among formerly independent elements. We present a programming model that alleviates this issue by turning interaction adaptations into first-class citizens independent of the objects they orchestrate. This model is based on *co-occurrences*, the prerequisite conditions for interactions to take place, and *entanglements*, the temporary coupling and interplay among elements of an interaction. We describe Tangler, a proof-of-concept implementation of this model, and demonstrate its power and generality with two adaptation scenarios.

CCS Concepts: • **Human-centered computing → HCI theory, concepts and models**; **User interface programming**.

Additional Key Words and Phrases: Interaction Architectures, Malleable Software, Tailorability.

ACM Reference Format:
Anonymous Author(s). 2018. Co-Occurrences and Entanglements: A Programming Model for Open-Ended Interactive Software. In *EICS '20: ACM Symposium on Engineering Interactive Computing Systems, June 23–26, 2020, Sophia Antipolis, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

0.1 One to Two Mice

The task is to adapt a Tangler instance that is configured to use a single mouse cursor (as in any modern operating system) so that it instead has uses a different cursor for each mouse plugged into the computer. Before the adaptation, a single cursor will no matter how many mice are plugged in, and the movements of each mouse will be mapped to that cursor (figure 1, left). The adaptation must provide the ability to map each mouse to a different cursor, creating and destroying the cursors as the mice are plugged in and unplugged (figure 1, right).

Listing 1 shows the code implementing the initial mouse policy. defs.mouseEntanglement defines the cursor, its location, and the means of connecting each mouse to the cursor.

The key to this adaptation is to replace it with a component that can dynamically instantiate different cursor configurations depending on the particular mouse that it is created for. To accomplish this, we create a record that maps a mouse identifier, which identifies the hardware model, to a type defining the appropriate configuration (listing 2).

The last step is to add a component to the mouse entangler that replaces the old entanglement with the new one (listing 3).

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

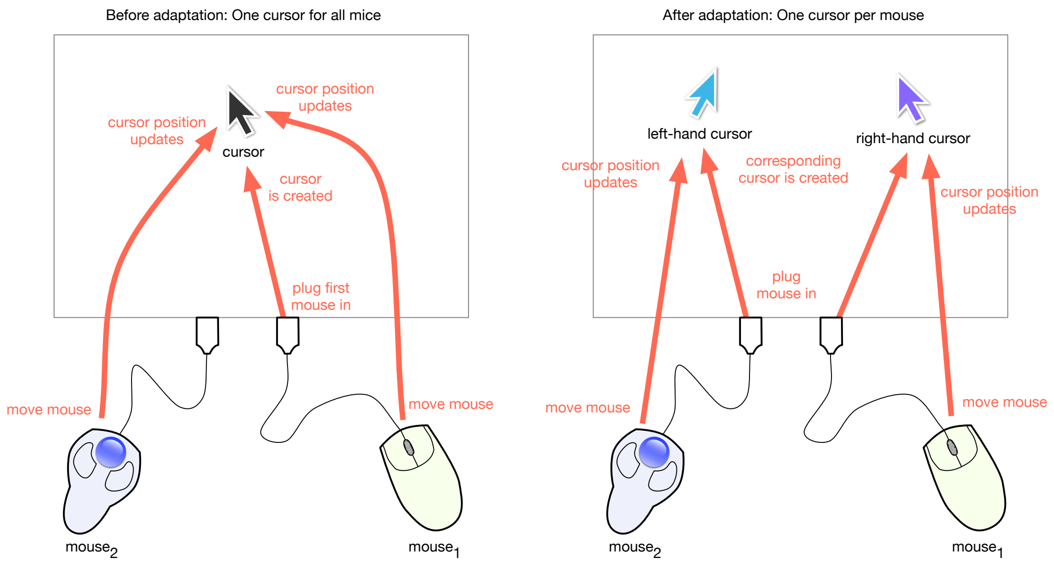

Fig. 1. The behavior associating mice and cursors before and after the desired adaptation.

0.1.1 Analysis. This example illustrates the generality and power of our model. In particular, it demonstrates how entanglers can extend the space of variation of an interactive system in a robust way: the result of this adaptation is a system that provides a meaningful interface whether there are zero or more mice plugged in.

In the above example, defs.rightHandCursorConfiguration applies the same cursor and transducer types to each new mouse as the original setup. In practice it would likely be customized, at a minimum to visually distinguish the right- and left-hand cursors. This would not have to happen immediately however, as the new cursors may be customized independently by distributing further options to them.

```
tangler.define('defs.cursor', {
  parentType: 'tangler.viewComponent',
  model: {
    position: {
      x: 0,
      y: 0
    },
    radius: '10 px'
  },
  draw: 'defs.drawCursor'
});

tangler.define('defs.mouseEntangler', {
  parentType: 'tangler.entangler',
  conditions: [
    'mouse matches :mouse'
  ],
  entanglementTemplate: {
    name: 'mouseExists',
    type: 'defs.mouseEntanglement',
    options: {
      mouse: '{source}.mouse',
    }
  }
});

tangler.define('defs.mouseEntanglement', {
  parentType: 'tangler.coOccurrence',
  distributeOptions: {
    // create a cursor at the constant location {ui}.cursor
    // '{ui}' resolves to a component under the root component,
    // which contains all elements in the "interface" layer.
    // redundant instances of this distribution have no effect
    createCursor: {
      record: 'defs.cursor',
      target: '{ui}.options.components.cursor.type'
    }
  },
  components: {
    // mirror the created cursor locally
    cursor: '{ui}.cursor',
    // create a transducer connecting
    // the detected mouse and created cursor
    transducer: {
      type: 'defs.mouseCursorTransducer',
      options: {
        components: {
          mouse: '{that}.mouse',
          cursor: '{ui}.cursor'
        }
      }
    }
  }
});

tangler.define('defs.mouseCursorTransducer', {
  parentType: 'tangler.component',
  distributOptions: {
    forwardMouseEvents: {
      record: {
        'device-mousedown': '{coOccurrence}.cursor.events.device
       -mousedown',
        // ... additional forwarded events omitted ...
      },
      target: '{coOccurrence}.cursor.events'
    }
  },
  modelRelays: {
    // cursor.position = cursor.position + mouse.delta
    // rules using un-invertible transforms work one way
    cursorAccumulatesMouse: {
      target: '{that}.cursor.model.position',
      transform: {
        x: {
          transformType: 'tangler.add',
          left: '{that}.mouse.model.delta.x',
          right: '{that}.cursor.model.position.x'
        },
        y: {
          transformType: 'tangler.add',
          left: '{that}.mouse.model.delta.y',
          right: '{that}.cursor.model.position.y'
        }
      }
    }
  }
});
```

Listing 1. The initial mouse policy

```
tangler.define('defs.bimanualMouseEntanglement', {
  parentType: 'tangler.coOccurrence',
  mouseToCursorMap: {
    'mouse-1133-49232': 'defs.rightHandCursorConfiguration',
    'mouse-1149-4099': 'defs.leftHandCursorConfiguration',
    // 'default' value is returned for invalid keys
    // this creates a ``catchall'' cursor for additional mice
    'default': 'defs.genericCursorConfiguration'
  },
  components: {
    cursorConfiguration: {
      // rather than provide a static type name here,
      // resolve one dynamically based on the mouse id.
      // the array-form path avoids ambiguous parsing
      type: ['{that}', 'mouseToCursorMap', '{that}.mouse.id'],
      options: {
        components: {
          mouse: '{that}.mouse'
        }
      }
    }
  }
});

tangler.define('defs.rightHandCursorConfiguration', {
  parentType: 'tangler.component',
  distributeOptions: {
    createCursorRemotely: {
      record: 'defs.cursor',
      target: '{ui}.options.components.rightHandCursor.type'
    },
    mirrorCursorLocally: {
      record: '{ui}.rightHandCursor',
      target: '{that}.options.components.cursor'
    }
  },
  components: {
    transducer: {
      type: 'defs.mouseCursorTransducer',
      options: {
        mouse: '{that}.mouse',
        cursor: '{that}.cursor',
      }
    }
  }
});

// ... Other cursor configurations omitted ...
```

Listing 2. The mouse entangler

```
tangler.define('defs.bimanualMousePolicyDistributor', {
  parentType: 'tangler.component',
  distributeOptions: {
    setMousePolicy: {
      record: 'defs.bimanualMouseCoOccurrence',
      target: '{mouseEntangler}.entanglementTemplate.type'
    }
  }
});

tangler.make('root.mouseEntangler.options.components.mousePolicy', 'defs.bimanualMousePolicyDistributor');
```

Listing 3. The mouse distributor