TurTan: Un Lenguaje de Programación Tangible Para el Aprendizaje

Carles F. Julià, Daniel Gallardo, and Sergi Jordà

Universitat Pompeu Fabra
{carles.fernandez, daniel.gallardo, sergi.jorda}@upf.edu

Resumen En este artículo se presenta la segunda iteración de TurTan un lenguaje de programación tangible basado en Logo. Esta segunda iteración se caracteriza por la inclusión de dos nuevos elementos: tarjetas y modificadores; añadidos a los objetos anteriores que formaban el grupo de instrucciones.
También se repasa la interacción hablando de la experiencia del usuario, vinculación de instrucciones, mapeo de ángulos y capas de interacción. Y finalmente se presentan una serie de programas de muestra y la proyección del trabajo futuro con este lenguaje de programación.

1. Introducción

La gran mayoría de los lenguajes de programación son textuales, lo cual los hace propensos a errores ortográficos y de sintaxis, exigiendo de sus usuarios un incremento de habilidades y conocimientos previos para su uso. Por otro lado, aunque en menor medida, existen diferentes tipos de lenguajes de programación gráficos basados en la vinculación visual y conceptual de módulos o elementos. Estos lenguajes, que por su propia naturaleza, están a menudo exentos de estos tipos de errores sintácticos u ortográficos, pueden constituir un buen punto de partida para el aprendizaje de los conceptos básicos de la programación.

Las interfaces tangibles, constituyen otra forma de comunicarnos con los ordenadores. A diferencia del modelo de comunicación actual (GUI), las interfaces tangibles(TUI) permiten al usuario tocar directamente la información con sus manos, manipular literalmente los datos. Estas interfaces, se caracterizan por no intimidar a los usuarios no expertos así como promover actividades exploratorias, expresivas y experimentales[6]. Además de los affordances que ofrecen este tipo de interfaces y particularmente combinadas con superficies tangibles, animan a la interacción social y a la colaboración[4].

Intentando aprovechar estas ventajas para crear un lenguaje de programación orientado al aprendizaje de los conceptos básicos de la computación, hemos creado TurTan[2]. Un lenguaje inspirado en Logo[1] que hace uso de objetos a modo de instrucciones, que se edita y ejecuta sobre la superficie de una mesa interactiva y permitie la creación de cadenas de instrucciones mediante la vinculación ordenada de estos objetos.

2. Trabajos relacionados

Puesto que este proyecto hereda varios conceptos de la interacción con interfaces tangibles y de los lenguajes de programación orientados al aprendizaje, los trabajos relacionados con éste son muchos y variados. Por un lado disponemos de múltiples ejemplos de interfaces tangibles ya sean basadas en objetos o en mesas retro-proyectadas. Por el otro lado, disponemos de varios lenguajes de programación orientados a un publico infantil como Logo (en el cual se inspira la figura de la tortuga en este proyecto) o lenguajes visuales orientados a usuarios más expertos como podrían ser PureData o MaxMSP.

2.1. Interfaces de usuario tangibles

Las interfaces de usuario tangibles se caracterizan por facilitar la manipulación directa de los datos mediante piezas y mecanismos tangibles; haciendo tangible lo intangible.

Este tipo de interfaces, se están haciendo cada vez más populares académicamente con la celebración de diversas conferencias especializadas (TEI, tabletop) o incluso a nivel comercial (Reactable[5], Percusa AudioCubes[8], NuiGroup) donde se pondrá a prueba al usuario final.

Existen varios diseños de interfaces tangibles que intentan solucionar o facilitar diferentes actividades. Estas son tan dispares como el MARBLE ANSWERING MACHINE de Durrell Bishop[7], primer ejemplo de interfaz tangible donde se pretendía crear un contestador automático el el cual los mensajes estuviesen representados por canicas, o incluso hasta ejemplos de programación tangible como AlgoBlocks.

La programación tangible, no implica la colocación de una instrucción encima de otra. En el caso de Quetzal[3], los programadores parten de una serie de cartoncillos que van poniendo uno al lado de otro para generar la cadena de instrucciones. Estos cartoncillos, están impresos y recortados de tal manera que su forma y colores indican el tipo de instrucción y permiten ,posteriormente ser fotografiados para que un ordenador interprete las cadenas o estructuras hechas con ellos.

Siguiendo en la línea de Quetzal, encontramos AlgoBlocks. Un sistema de mini-cubos tipo "LEGO" que se encajan unos con otros para crear cadenas o bloques de instrucciones (programas). La peculiaridad de Algoblocks es que cada pieza representa a una instrucción determinada y ofrece el control del parámetro asociado a esta mediante diferentes botones o sensores integrados en la misma pieza. El objetivo de AlgoBlocks es crear una série de estructuras lógicas (programas) mediante el ensamblado de diferentes piezas para "construir" aplicaciones de forma tangible. La diferencia respeto Quetzal, recae en el compilador, mientras que en AlgoBlocks los bloques se interpretan en *tiempo de programación*, en Qetzal se pasa por el proceso de fotografiar la cadena y procesarla posteriormente, hecho que da ventaja a Algoblocs ya que permite saber en todo momento el resultado del programa y evitar así, errores de sintaxis.

Finalmente, aportando todavía más dinamismo a la programación, podemos encontrar Reactable, un ejemplo claro de manipulación directa de la información en tiempo real. Reactable funciona en una superficie tangible multi-táctil donde el usuario mediante sus dedos y una serie de bloques de plástico se comunica con el sistema que dibuja en la superficie el feedback visual (resultado) procedente de dicha interacción. El objetivo es generar estructuras con las diferentes piezas para generar sonidos y ritmos distintos. Estas piezas, se vinculan automáticamente según proximidad y evitan así todo tipo de errores sintácticos ya que si dos piezas no pueden relacionarse, no se vincularan nunca.

2.2. Lenguajes de programación orientados al aprendizaje

Este tipo de lenguajes de programación se caracterizan por tener una sintaxis simple, resultados visuales y curvas de aprendizaje muy rápidas.

Entre estos lenguajes, podemos encontrar Logo un lenguaje basado en la escritura con instrucciones muy fáciles de aprender y , en algunas versiones, traducidas a varias lenguas (entre ellas el español). El resultado de cada instrucción, se ve plasmado en el recorrido efectuado por un "robot" (popularmente una tortuga virtual). Actualmente es el lenguaje más utilizado para enseñar la mayoría de los principales conceptos de la programación.

Por otro lado, tenemos Starlogo, un lenguaje de programación de videojuegos en tres dimensiones cuya sintaxis se basa en enlazar virtualmente varias cajas o bloques creando diferentes secuencias que definen el comportamiento de los objetos en el juego (enemigos, plataformas, vehículos,...). Este tipo de lenguajes, tienen la ventaja de que están exentos de errores de sintaxis y semántica ya que las diferentes cajas representan las instrucciones(sintaxis) y cada caja solo se puede enlazar con un tipo de cajas predeterminado (metáfora de puzzle) solucionando errores semánticos.

Finalmente, podemos encontrar lenguajes de programación encubiertos en juegos. Como en el caso de Little Big Planet (Playstation 3), un potente emulador de física donde los jugadores pueden definir sus propios escenarios mediante la definición de características físicas de materiales (cuerdas, muelles, superficies viscosas, accionadores y actuadores) programando de esta manera diferentes objetos.

3. Introduciendo TurTan

Para poder crear programas y ejecutarlos necesitamos de tres herramientas distintas: Un lenguaje de programación, un editor de código fuente y un intérprete (o un compilador). TurTan reúne estas tres herramientas en un solo entorno autosuficiente.

TurTan es un lenguaje de programación orientado a dibujo vectorial del tipo Logo. Tiene instrucciones , parámetros y funciones.

También es un editor de programas hechos en TurTan(lenguaje). Las instrucciones son objetos tangibles que, colocados sobre una mesa interactiva, definen el comportamiento del programa.

TurTan se encarga también de interpretar código creado en TurTan(editor). Este ejecuta el programa y dibuja el resultado en la mesa interactiva, permitiendo su exploración y visualización con gestos hechos con los dedos.

Aún reuniendo estas tres características (lenguaje, editor e intérprete) TurTan no pretende ser una forma seria de crear programas. No se pueden hacer sistemas operativos o cálculos precisos así como crear cualquier tipo de programa que no sea puramente de dibujo, y aún siendo así, no puede crear todos los dibujos posibles.

La función de TurTan es la de enseñar conceptos de la programación a no programadores. Alejarse de la estricta visión de la programación textual que requiere de habilidades concretas y poco comunes como la mecanografía, la sintaxis compleja o la memoria abstracta y centrarse en los conceptos propios de ésta: las instrucciones, los procedimientos, la recursividad... y enfocarlos de una forma entretenida y gráfica.

Como en una aplicación tangible, TurTan no distingue entre el programa y la interfaz: cada objeto es a la vez real y virtual.

4. La sintaxis

Siendo TurTan un lenguaje de programación, tiene una gramática concreta y definida. Ésta define las relaciones que se pueden dar entre los 3 tipos de objetos que se pueden usar en la mesa: Las instrucciones, los modificadores y las tarjetas. Para ver ejemplos de programas reales vean la sección 6.

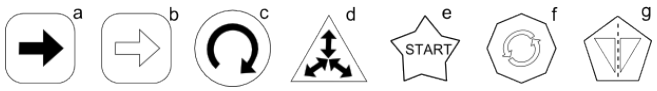
4.1. Instrucciones

Esta es la parte mas parecida a los lenguajes de programación escritos: cada objeto representa una instrucción concreta que realizará la tortuga. Una vez realizada esta acción la tortuga por norma general se quedará en la posición en la que se encuentra y realizará la siguiente.

Los objetos se vinculan entre ellos como se describe en la sección 5.2, cualquier otro movimiento de éstos en X e Y no afectará al sistema. Cada instrucción, contiene un parámetro que puede ser modificado por el usuario para ajustar la distancia, rotación, repeticiones u otras variables dependiendo de la instrucción. Para modificar este parámetro solo es necesario hacer rotar el objeto sobre si mismo y esta acción será mapeada (Sección 5.4) sobre el parámetro de la instrucción.

Para crear una cadena de instrucciones partiendo de la mesa vacía, basta con introducir objetos secuencialmente uno al lado del otro donde el sistema se encarga de dibujar las relaciones entre ellos. El orden de ejecución, es definido según el orden de vinculación de la pieza en la cadena de instrucciones y este no se puede deshacer si no es quitando el objeto instrucción de la mesa. Llamamos a este tipo de comportamiento, vinculación estática (Sección 5.2).

TurTan parte de 7 instrucciones básicas que son las siguientes:



a) Avanzar pintando Hace avanzar la tortuga x posiciones pintando.
b) Avanzar sin pintar Hace avanzar la tortuga x posiciones sin pintar.
c) Girar Hace rotar la tortuga x grados.
d) Escalar Modifica x veces el tamaño de la tortuga y futuras instrucciones.
e) Volver al inicio Resetea la posición de la tortuga a la inicial.
f) Repetir Repite x veces todas las instrucciones des del inicio.
g) Reflejar Invierte los ángulos, efecto espejo o simetría.

4.2. El metalenguaje

Ahora introducimos los modificadores, nuevos en esta segunda versión de TurTan. Los modificadores son objetos que no representan una instrucción ni un trozo de código, sino que representan una función modificadora del parámetro de una instrucción.

Estos modificadores, suelen ser funciones que según el instante de tiempo modifican el parámetro de la instrucción. Este concepto lo podemos encontrar en proyectos anteriores como Reactable donde se usan para generar melodías, modular frecuencias, etc. En esencia, en TurTan es lo mismo, aunque en nuestro caso nos permite crear animaciones en lugar de modificar sonidos.

A diferencia de las instrucciones, los modificadores se vinculan a los otros objetos en tiempo real [Sección5.2]. Eso quiere decir que al mover el modificador sobre la mesa el efecto del modificador puede recaer en diferentes piezas en momentos distintos.

Actualmente, en oposición a las instrucciones, no existe un conjunto cerrado de modificadores (osciladores, lineales, logarítmicos,....) sino que estos están pensados en ser ampliados según las necesidades concretas de cada ocasión. Por ejemplo para programar visuales de música se pueden añadir modificadores que vayan acorde con el BPM de la canción o añadir controles externos como cada vez que se pulse un botón, incremente en x el parámetro de una instrucción.

Finalmente, para concluir este apartado, es necesario mencionar los grados de libertad de los modificadores. Esto nos dirá de cuántas variables de entrada disponemos para cada modificador. Estas son básicamente 3: ángulo, distancia y rotación. Partiendo de la instrucción a la que se está aplicando la modificación hasta el modificador, encontramos el ángulo y la distancia entre estos dos objetos, la rotación es simplemente la rotación del modificador sobre sí mismo.

4.3. Tarjetas

Las tarjetas, tambien introducidas en la segunda versión de TurTan, son una serie de objetos hechos de cartón plastificado que sirven para almacenar y lanzar programas almacenados en su interior. El concepto de usar objetos como transporte de datos fue introducido en mediablocks[9], con esto se consigue dotar al sistema de elementos persistentes durante el tiempo y promueve el intercambio

de programas hechos con TurTan. Haciendo homología a los lenguajes de programación actuales, una tarjeta sería el programa ya compilado y listo para ser ejecutado en cualquier otro TurTan.

Tarjetas como depósitos La primera aplicación de las tarjetas es la de guardar programas de TurTan. Cada tarjeta puede contener un solo programa y éste puede ser asignado dinámicamente, copiando todo lo que hay en la mesa. Para hacerlo, el programador debe poner la tarjeta sobre la mesa, lejos de la cadena de instrucciones y desbloquearla efectuando una rotación de 180 grados sobre la tarjeta. Al efectuarse la grabación del contenido de la mesa, se mostrará como una copia de las formas de las instrucciones y el dibujo resultado se encoje y se introduce virtualmente dentro de la tarjeta. Una vez mostrado este efecto, aparecerá una vista previa del programa y del dibujo al lado de ésta. Estas tarjetas al ser plastificadas, se puede escribir tantas veces se como se quiera en su dorso el nombre del programa con un rotulador (por ejemplo "dibujo de un sol").

Para ver el contenido de las tarjetas del que se desconoce el interior, solo basta con situarlas en la mesa lo suficientemente lejos de la cadena de instrucciones(para evitar lincarla accidentalmente) y el usuario podrá observar una miniatura de lo que hay almacenado en cada tarjeta.

Tarjetas como instrucciones (subrutinas) La segunda aplicación de las tarjetas es también la más interesante. Las tarjetas, una vez llenas, pueden usarse como instrucciones: al ponerlas en la mesa cerca de la cadena de instrucciones estas se comportaran de igual forma enlazándose a la cadena y ejecutando el programa guardado. De hecho se comporta como una función sin parámetros (procedimiento). Imaginemos que dibujamos un árbol que se balancea, guardamos este programa en una tarjeta y despejamos la mesa. Ahora dibujamos una montaña, y usando la tarjeta "arbol" dentro de nuestra secuencia de instrucciones, podemos crear un bosque que se balancea.

Un punto importante a tener en cuenta cuando se crean tarjetas para ser usadas como subrutinas es que después de ejecutarlas desde la tarjeta, el estado de la tortuga se ve alterado. Por ejemplo si una tarjeta contiene un programa compuesto por una sola instrucción Girar, al ejecutarla desde la tarjeta todas las instrucciones siguientes del programa quedaran giradas. Para corregir este efecto y devolver la tortuga a su estado anterior existe la instrucción volver al inicio. Al cerrar las subrutinas dentro de las tarjetas con una instrucción Volver al inicio, permitimos poder llamar a estas subrutinas sin alterar el estado de la tortuga en la cadena del programa actual.

5. Interacción

5.1. Experiencia del usuario

El programador de TurTan[1],al empezar, se encuentra delante de una mesa interactiva vacía, con solo una tortuga dibujada en el medio. Alrededor de esta

[1] al ser TurTan un lenguaje de programación hablaremos de programador en vez de usuario, aunque sean intercambiables en este contexto.

mesa, y fuera de la superficie interactiva, hay varios objetos transparentes de diferentes formas, con símbolos en una cara y fiduciales en la otra.

Para programar en TurTan, tendrá que construir una estructura mediante las piezas sobre la mesa; colocándolas en fila representando el flujo del programa. Instantáneamente y de forma continua mientras el programador manipula el programa, el resultado de éste será dibujado en la mesa. No le serán necesarios pasos de compilación o ejecución: el resultado instantáneo, la representación de la salida del programa, estará allí presente.

Cuando el resultado del dibujo le plazca, el Programador podrá guardar el programa en una tarjeta con un fiducial, para verlo y usarlo mas tarde en sus propios programas en el futuro.

Finalmente en cuanto todas las instrucciones sean retiradas de la mesa interactiva, TurTan volverá a su estado original, con la tortuga en el medio.

En todo momento, mediante gestos, el Programador podrá inspeccionar con toda libertad el resultado del programa: ampliándolo, reduciéndolo, girándolo o trasladándolo.

5.2. Vinculación estática y dinámica

En TurTan, cuando un objeto es puesto sobre la mesa, este tiene que ser insertado en la cadena de instrucciones dependiendo de la proximidad con sus vecinos. Si por lo contrario la distancia con cualquier otro objeto es demasiado grande, este no se inserta en la cadena.

Una vez tenemos un objeto vinculado en la cadena y lo movemos, se pueden tomar dos acciones diferentes:

Vinculación dinámica: es aquella vinculación que puede ser rota cuando un objeto se mueve.

Vinculación estática: es aquella que nunca se modificara aunque los objetos se desplacen al otro lado de la mesa o estén desordenados a no ser que los quitemos de la mesa.

Mientras que la vinculación dinámica nos permite variar el orden de las instrucciones dentro de la cadena fácilmente. Para TurTan hemos decidido usar la vinculación estática ya que cumple una serie de condiciones que la hacen más adecuada para un lenguaje de programación:

- En la vinculación dinámica el orden no es importante, en la estática si.
- La vinculación estática nos permite mover la cadena de instrucciones sin "peligro" a cualquier lugar de la mesa.
- Si tenemos objetos muy cercanos entre si, con la vinculación dinámica podrían variar su orden sin el consentimiento del usuario.

5.3. Capas de interacción

TurTan consiste de dos espacios de interacción distintos: el editor de programa y la salida del programa. Ambos comparten el espacio de proyección de la

imagen y el área de interacción de la mesa. No hay ninguna división espacial entre las dos partes, la única división es conceptual en niveles de interacción: el editor se controla mediante objetos y la salida mediante los dedos.

Esta distinción es muy útil y efectiva. El programador nunca tendrá la duda de dónde se dirigirá una acción concreta sobre la mesa. Asimismo no hace falta crear áreas de delimitación en la interacción: no hay paneles, botones u otros elementos diseñados para la partición espacial de la interacción.

Esta forma de separación, aunque útil y efectiva, no es muy utilizada en la creación de superficies interactivas, muchas veces dominadas por los dedos solamente, y no siempre multi-táctiles. Posiblemente a causa de las influencias de WIMP y por tanto de la dictadura del ratón en la mente de los diseñadores de interfaces.

La interacción en la capa de la salida del programa utiliza movimientos de dedos para trasladar, rotar y escalar el dibujo creado por la tortuga. Con un solo dedo podemos trasladar todo el dibujo sin cambiar su tamaño u orientación: al arrastrar el dedo por la mesa, el punto del dibujo situado bajo el dedo tendera a mantenerse siempre debajo de él. De forma análoga a la anterior, utilizando dos dedos, la imagen deberá rotarse, trasladarse y escalarse para mantener los puntos asociados directamente bajo los dedos respectivos. Este tipo de manipulación de espacios es muy intuitiva y suele implementarse en cualquier tipo de aplicación multi-táctil de una forma u otra.

La interacción en la capa de edición ya ha sido bastante explicada en secciones anteriores. Los objetos, al ponerse sobre la mesa, reciben un aura de color que permite confirmar su correcto reconocimiento por parte del sistema. También automáticamente se conectan las piezas, mediante líneas continuas, usando un algoritmo de vinculación estática. Una propiedad que merece ser explicada mas a fondo es el mapeo de los ángulos de las instrucciones en sus parámetros, esencial para el buen control del programa.

5.4. Mapeo de ángulos

Al mapear los ángulos a parámetros, se tuvieron que tomar varias decisiones en el diseño, dada la naturaleza diversa de los parámetros de las instrucciones: inexistentes, continuos, infinitos, acotados, enteros, binarios...

En primer lugar tenemos que tomar la decisión que el ángulo absoluto no es importante. Dado que todos los parámetros pueden tener un valor inicial y que las piezas no dan ningún tipo de información de orientación (debemos pensar que la superficie es redonda y que por lo tanto cualquier indicación de orientación en las piezas sería fútil) es poco útil tener en cuenta su ángulo absoluto. Así el valor básico usado para el mapeo será el ángulo relativo más que el ángulo absoluto.

Siguiendo esta primera aproximación pudimos mapear los incrementos en el ángulo del objeto, $\Delta\alpha$, en incrementos en el parámetro de la instrucción, $\Delta\beta$. Podemos notar esta relación de la forma siguiente:

$$\Delta\beta = k\Delta\alpha$$

Esta forma de mapeo demostró tener bastantes problemas a la hora de modificar parámetros en ciertas situaciones. Por ejemplo, dentro de un bucle una instrucción de girar puede tener un gran impacto sobre el resultado final: el programador necesita una gran precisión. Por otro lado fuera de un contexto de repetición, el giro deseado puede ser extenso. Para poder conseguir esta versatilidad debemos hacer una transformación no lineal $f(x)$ de los incrementos del parámetro que permitan que manteniendo que $sign(\Delta\alpha) = sign(\Delta\beta)$, para valores pequeños de $\Delta\alpha$ se cumpla $f(\Delta\beta) < k\Delta\beta$ y para valores grandes de $\Delta\alpha$ se cumpla $f(\Delta\beta) > k\Delta\beta$. Esta función $f(x)$ podría ser:

$$\Delta\beta = sign(\Delta\beta)\frac{\Delta\alpha^2}{t}$$

donde t es el valor de $\Delta\alpha$ donde $\Delta\beta = \Delta\alpha$, y nos indica donde cambia el comportamiento de precisión a potencia.

6. Ejemplos de programas

[elemento cortado, para artículo completo, mandar email al autor]

7. Trabajo futuro

Esta no será la última iteración de TurTan. Aún hay problemas que resolver y oportunidades de aprovechar para ampliar sus capacidades.

Edición de programas guardados Hemos introducido un método para guardar programas y subrutinas, pero una vez guardada en una tarjeta, una secuencia ya no puede ser modificada. Debemos encontrar una solución a este problema de la forma mas intuitiva y coherente que sea posible.

Edición de modificadores Ahora que hemos introducido el concepto de modificador, debemos inventar un sistema fácil para aprovecharlo. Quizás no baste en crear diferentes tipos de osciladores, poder extender los modificadores para añadir comportamientos complejos por parte de los usuarios sería muy potente y útil.

En educación Aunque presentamos TurTan como herramienta para poder aprender conceptos básicos de la programación, no se han hecho pruebas formales al respecto para confirmar su utilidad en este aspecto.

En vídeo-jockey Una de las posibilidades que se abren con la personalización de los modificadores podría ser la fácil creación de dibujos animados al ritmo de una música por ejemplo. Siendo los dibujos de TurTan visualmente agradables, y siendo TurTan una aplicación de dibujo y animación en tiempo real creemos que una buena idea utilizarlo para vídeo-jockey.

Referencias

1. W. Feurzeig, S. Papert, Mo Bloom, R. Grant, and C. Solomon. Programming-languages as a conceptual framework for teaching mathematics. In *the National Science Foundation*, 1969.
2. D. Gallardo, C.F. Julia, and S. Jorda. TurTan: A tangible programming language for creative exploration. In *3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008*, pages 89–92, 2008.
3. M.S. Horn and R.J.K. Jacob. Designing tangible programming languages for classroom use. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 159–162. ACM New York, NY, USA, 2007.
4. Hornecker, Eva and Buur, Jacob. Getting a grip on tangible interaction: a framework on physical space and social interaction. In *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems*, volume 1 of *Designing for tangible interactions*, pages 437–446, 2006.
5. Sergi Jordà, Gunter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reactable: exploring the synergy between live music performance and tabletop tangible interfaces. In Brygg Ullmer and Albrecht Schmidt, editors, *Tangible and Embedded Interaction*, pages 139–146. ACM, 2007.
6. Yvonne Rogers Paul Marshall and Eva Hornecker. Are tangible interfaces really any better than other kinds of interfaces? In *Pervasive Interaction Labthe*, Open University, Milton Keynes, MK7 6AA, UK, 2007.
7. R. Poynor. The hand that rocks the cradle. *ID-The International Design Magazine*, 1995.
8. B. Schiettecatte and J. Vanderdonckt. AudioCubes: a distributed cube tangible interface based on interaction range for sound design. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 3–10. ACM New York, NY, USA, 2008.
9. B. Ullmer, H. Ishii, and D. Glas. mediaBlocks: physical containers, transports, and controls for online media. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 379–386. ACM New York, NY, USA, 1998.