Hasselt:
Rapid Prototyping of Multimodal Interactions with Composite Event-Driven Programming

Fredy Cuenca, School of Mathematical Sciences and Information Technology, Yachay Tech, San Miguel de Urcuquí, Ecuador & Expertise Centre for Digital Media, Hasselt University – tUL – imec, Diepenbeek, Belgium
Jan Van den Bergh, Expertise Centre for Digital Media, Hasselt University – tUL – imec, Diepenbeek, Belgium
Kris Luyten, Expertise Centre for Digital Media, Hasselt University – tUL – imec, Diepenbeek, Belgium
Karin Coninx, Expertise Centre for Digital Media, Hasselt University – tUL – imec, Diepenbeek, Belgium

ABSTRACT
Implementing multimodal interactions with event-driven languages results in a ‘callback soup’, a source code littered with a multitude of flags that have to be maintained in a self-consistent manner and across different event handlers. Prototyping multimodal interactions adds to the complexity and error sensitivity, since the program code has to be refined iteratively as developers explore different possibilities and solutions. The authors present a declarative language for rapid prototyping multimodal interactions: Hasselt permits declaring composite events, sets of events that are logically related because of the interaction they support, that can be easily bound to dedicated event handlers for separate interactions. The authors’ approach allows the description of multimodal interactions at a higher level of abstraction than event languages, which saves developers from dealing with the typical ‘callback soup’ thereby resulting in a gain in programming efficiency and a reduction in errors when writing event handling code. They compared Hasselt with using a traditional programming language with strong support for events in a study with 12 participants each having a solid background in software development. When performing equivalent modifications to a multimodal interaction, the use of Hasselt leads to higher completion rates, lower completion times, and less code testing than when using a mainstream event-driven language.

KEYWORDS
Composite Events, Declarative Languages, Event Languages, Event-Driven Programming, Interactive Systems, Multimodal Systems, Rapid Prototyping

INTRODUCTION
Rapid prototyping multimodal interactive systems consists of implementing, evaluating, and refining different types of multimodal interactions in an iterative fashion. These progressive refinements enable developers to gain a proper understanding of the strengths and weaknesses of different possible solutions. They arrive at a set of interactions that need to be supported by the final system. Rapid
prototyping must be inexpensive in effort, since the goal is to quickly explore a wide variety of possible
types of interaction. This involves building, evaluating, and throwing away many prototypes without
remorse (Beaudouin-Lafon, 2003). In the remainder of this article we use the term developers to
indicate developers of multimodal interactive systems that participate in rapid prototyping activities.

It is commonly accepted that the event-driven paradigm is a good match for realizing the
implementation of interactive systems (Lewis & Rieman, 1993). However, in the case of multimodal
interactive systems, the use of this paradigm may adversely affect the speed and cost of the rapid
prototyping phase significantly. When implementing multimodal interactions, the usage of event-
driven languages results in code that is dedicated in large part to the management of the interaction
state. This code is then plagued with a multitude of flags that developers have to update in a self-
consistent manner and across different event handlers (Spano, Cisternino, Paternò, & Fenu, 2013;
Kin, Hartmann, DeRose, & Agrawala, 2012; Cuenca, Van den Bergh, Luyten, & Coninx, 2014). The
resulting ‘callback soup’ makes it difficult to understand and to change the multimodal system source
code. This complexity has to be faced for each iteration of the prototyping phase.

Several (mostly visual) languages have been proposed with the aims of facilitating the creation
of multimodal prototypes (Bourguet, 2002; Dragicevic & Fekete, 2004; De Boeck, Vanacken,
Raymaekers, & Coninx, 2007; Lawson, Al-Akkad, Vanderdonckt, & Macq, 2009; Navarre, Palanque,
Ladry, & Barbouli, 2009; König, Rädle, & Reiterer, 2010; Hoste, Dumas, & Signer, 2011; Dumas,
Signer, & Lalanne, 2014). These languages allow the developer to describe multimodal interactions
at a high-level of abstraction bypassing the need to manually maintain the interaction state, as it is
needed with event-driven languages. To a greater or lesser extent, the aforementioned languages have
accomplished their main goal of simplifying the creation of multimodal prototypes. Despite this, for
many of these languages abstraction also means giving up the fine-grained control when dealing with
events directly. In other words, these approaches dismiss the programming experience of developers
and replace this with some formalism that hides details and introduces a more abstract terminology.
Abstraction by means of visual models may not be the method of choice for many developers, who,
instead, use textual languages or at least access and modify the code that drives the interactive
system. Since familiarity with a language is an important factor that has a strong, positive influence
in programming language adoption (Meyerovich & Rabkin, 2013), we created a language that saves
developers from dealing with the ‘callback soup’ problem, while building upon familiar concepts
and well-known programming practices.

Hasselt is a textual, declarative language that allows the description of executable multimodal
interaction models. The core concept of Hasselt is a composite event, which is essentially a user-
defined sequence of events that are logically related (for example, because these are part of the same
interaction). Within Hasselt, developers define composite events by connecting several primitive
events (e.g. touch events or speech inputs) by means of specialized operators. Each operator represents
a specific relation between their operands. The overall composite event can then be bound to one or
more event handlers, which specify the behavior the system should expose when the composite event
occurs. At runtime, the event handlers are executed every time their associated composite events
occur. For event detection, Hasselt relies on existing recognizers to process the low-level input (like
speech, mid-air gestures or mouse movements) and does not replace existing recognition-based fusion
engines (D’Ulizia, 2009; Nigay & Coutaz, 1995; Bouchet, Nigay, & Ganille, 2004).

One can implement the “put-that-there” interaction (Bolt, 1980) —probably the best known
element of multimodal interaction— in Hasselt with a composite event, ptt (Figure 2) that combines
speech events and pointing events and specifies their temporal constraints (e.g. the pointing gestures
must be synchronized with the spoken pronouns ‘that’ and ‘there’ to avoid ambiguities). Such a
composite event can be bound to a function, putThatThere(), which will put the selected object at
the specified position once the interaction is completed (i.e. once ptt occurs). When desired, one can
also bind additional functions that are called before the interaction is completed (i.e. in response to
the partial detection of ptt), e.g. to highlight the object identified as ‘that’.
Success Factors in the Implementation of a Collaborative Technology, and Resulting Productivity Improvements in a Small Business: An Exploratory Study

www.igi-global.com/article/success-factors-implementation-collaborative-technology/3778?camid=4v1a