Programming Interactivity Requires Both Semantics and Semiotics

Joseph C. Osborn, Michael Mateas
University of California, Santa Cruz

Introduction. Programming languages are generally designed around models of computation, classes of programs, or populations of programmers. Considering interactive applications (e.g. games and single-page Web applications), we argue that programming language research could be usefully oriented around the end-user's experience of interacting with the system. *Operational logics*, a theoretical framework borrowed from the study of games and other interactive media, are an especially suitable formalism that accounts for both the subjective experience of users and the expressive requirements of program authors. While end-user experience is not a traditional subject of PL research, this move helps explain the recent success of the React user interface library and could yield future research directions.

Operational logics combine abstract operations with communicative strategies, at once describing both *how it works* and *what it looks like* [7, 13]. This contrasts with earlier approaches in game studies that split game rules from game fiction (art, sounds, text, and so on) and studied them separately; in fact, this was always a contested division [2, 10]. Animation and sound effects strongly influence how players perceive and employ a game's rules, and are in turn contextualized and given interpretations by their interplay with the rules.

A similar shift is happening in Web development today. React explicitly combines behavior and presentation in user interface components, de-emphasizing CSS in favor of styling and presentation via JavaScript because UI components are often *defined* by a tight coupling between visual appearance, underlying state, and time-evolving interactive behavior [3]. The behavior/presentation split impairs reasoning about the whole component, since each part of the specification implemented in multiple distinct locations and languages. Bret Victor's work is similar in spirit, advocating direct manipulation and reducing the distance between user-domain concepts and computation [12], echoing the principles of the Morphic user interface toolkit. We assert that the motivating pain in all these cases is the untenable separation of an interactive system into appearance and behavior: effectively the same split that caused so much strife and spilled ink in game studies.

We do not argue that *ad hoc* separation of concerns along the lines of UI components is ideal: we instead must identify a new principle for decomposing interactive applications. Besides the ubiquitous presentation/behavior split, concerns have been separated and individually addressed according to factors like performance or the expected authors of changes. We assert that fields which study interaction with computer programs and other rule-based systems could provide useful criteria for breaking down interactive programs into smaller parts. Here we consider *game studies*, which theorizes games, play, and players.

Operational Logics. Simulations have inspired PL research at least since *Simula*, but games are more than just simulations with user input. Players must form a mental model of the system with which they are interacting precisely enough to plan towards specific goals, revising this model as necessary [6]. This interplay between computational process and observed phenomena is captured by *operational logics*; each logic combines a set of *abstract operations* like those of collision detection or timed state machines with a *communicative strategy* such as 2D sprites or colored text.

For example, players interpret moving images as *solid simulated objects* because they halt when overlapping each other; they see an increasing number next to a small image of a hamburger as *gaining more food* because their own sprite has just touched a larger image of a hamburger. It matters that hamburgers are food, that the small icon and larger image match up, and that the counter increases (invoking abstract operations of a *resource logic*) due to player actions (in the *collision logic*). These interpretations involve both computation and presentation: if the player's sprite started reacting differently to collisions—if it occasionally moved through enemies unimpeded—but the appearance of this sprite did not change, players would be unable to predict or make sense of the new behavior. Most games are made up of several operational logics (*OLs*); each game rule is built from OLs' abstract operations and surfaced via their communicative strategies.

Game rules roughly correspond to program features, while games' observable phenomena correspond to user interface elements and feedback. Considering the audiovisual phenomena of a game apart from its rules is exactly as problematic as separating an application's presentation and behavior. The two are inextricably linked for the user and the designer, but in the source code the linkage is implicit, invisible, and tenuous.

An operational logics approach to programming emphasizes the reuse and composition of just a few OLs, with each such logic providing a few flexible operations and communicating via pre-determined channels (e.g. arcs and lines to be rendered by a drawing library). We thereby commit to a constrained language for expressing the required concepts, both from the standpoint of the developer (in terms of authorial affordances) and the interactor (in terms of interpretive affordances).

A music-playing application models a library of songs with membership in various playlists and other views. A *tabular logic* organizes each song's metadata into columns and rows, with the resizing of columns governed by a collision logic. Tabular logic's abstract operations come from relational algebra and its communicative strategy lays out each row and column in a grid. Though actually playing music requires calls to system libraries, the semantic connection between the UI's playhead and the song's current audio sample is maintained by an operational logic: the sound emitted by the speaker *communicates* the playhead's movement, just as arcs and lines render its visual appearance.

Games and Programming Languages. Previous presentations at this workshop have proposed games as a domain for programming language research [5]; we will further reinforce that argument. Game makers may be a friendlier audience for the claims and tools of programming language researchers than application developers in general [8]. Games have relatively explicit but frequently evolving designs, and some lightweight mathematical modeling techniques are already in use [11].

Many game-making tools already address particular compositions of operational logics: *Twine* [4] privileges linking logics, and *Idle Game Maker* [9] provides a language for resource logic games. Game developers have also been known to invent their own configuration or programming languages as needed. Games are therefore an ideal domain for exploring the design and composition of domain-specific languages (*DSLs*), and operational logics give a strong theoretical foundation.

We can also identify OLs with models of computation, immediately connecting to existing research communities. For example, *hybrid automata* are used to model systems with both discrete and real-time components [1]. They naturally describe the composition of *physics logics* and *character-state logics*; we can therefore use their verification and parameter synthesis tools directly on games made up of these logics.

Finally, most games are not capable of universal computation during play and should not require universal programming languages for their implementation. Games provide large, natural classes of non-universal machines, and we should seek convenient ways to specify such systems; universality is also unnecessary for many non-game interactive systems.

Conclusion. When building software for humans, we must account for users not just in our programming practices, but programming languages as well. The craft of making games presents an interesting and challenging domain for PL research; it could in turn benefit from the PL community's involvement.

References.

[1] Rajeev Alur et al. *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*. Springer, 1993.

[2] Gonzalo Frasca. "Ludologists love stories, too: notes from a debate that never took place". In: *International Conference of the Digital Games Research Association*. 2003.

[3] Pete Hunt. "React: Rethinking Best Practices". In: *JSConf EU 2013*. 2013. URL: `https://www.youtube.com/watch?v=x7cQ3mrcKaY`.

[4] Chris Klimas. *Twine*. 2009. URL: `http://twinery.org`.

[5] Chris Martens. "Languages for Computational Creativity: Generative Art and Virtual Worlds". In: *POPL Off the Beaten Track*. 2013.

[6] Michael Mateas. "Expressive AI: A Semiotic Analysis of Machinic Affordances". In: *Conference on Computational Semiotics for Games and New Media*. Conference on Computational Semiotics for Games and New Media. 2003.

[7] Michael Mateas and Noah Wardrip-Fruin. "Defining operational logics". In: *International Conference of the Digital Games Research Association*. 2009.

[8] Mark J Nelson and Michael Mateas. "A requirements analysis for videogame design support tools". In: *International Conference on Foundations of Digital Games*. ACM. 2009, pp. 137–144.

[9] Orteil and Opti. *Idle Game Maker*. 2014. URL: `http://orteil.dashnet.org/experiments/idlegamemaker/help`.

[10] Celia Pearce. "Theory Wars: An Argument Against Arguments in the so-called Ludology/Narratology Debate". In: *International Conference of the Digital Games Research Association*. 2005.

[11] Paul Tozour. "Decision Modeling and Optimization in Game Design". In: *Gamasutra* (2013).

[12] Bret Victor. *Drawing Dynamic Visualizations*. 2013. URL: `https://vimeo.com/66085662`.

[13] N. Wardrip-Fruin. "Playable media and textual instruments". In: *Dichtung Digital* 34 (2005), pp. 211–253.