CEL — Modeling Everywhere

Remo Lemma, Michele Lanza, Fernando Olivero
REVEAL @ Faculty of Informatics – University of Lugano, Switzerland

***Abstract*—The design of object-oriented systems starts with modeling, a process to identify core concepts and their relations. Mainstream modeling techniques can be either informal (white board, CRC cards, *etc.*) or formal (*e.g.,* UML editors). The former support well the creative modeling process, but their output is difficult to store, process and maintain. The latter reduce these problems, at the expense of creativity and productivity because they are tedious and not trivial to use.**

We present CEL, a touch- and gesture-based iPad application to rapidly create, manipulate, and store language agnostic object-oriented software models, based on a minimal set of constructs.

Demo video URL: `http://youtu.be/icQVS6w0jTE`.

I. INTRODUCTION

Software design is essentially modeling [1], a fundamental step in the software development process [2]. When performed incorrectly, this activity negatively influences the quality of the resulting system [3]. During the modeling phase, one tries to identify the core concepts of the system to be built. The produced model is a simplified representation of reality, which only includes information strictly serving the purpose at hand [4]. Modeling can be done in a *formal* or an *informal* way, each one with advantages and drawbacks.

Informal modeling includes methodologies such as paper sketches, whiteboard drawings, and CRC Cards [5]. It guarantees total freedom, but results in artifacts that are difficult to process, store, share and maintain. The maintenance problem is a crucial drawback, since design drift –a root cause of software aging [6]– is unavoidable in real-world systems [7]. Researchers have tried to improve these methodologies, because they are heavily employed to sketch potential solutions [8]. For example, electronic whiteboards have been leveraged in CALICO [9], and CREWW [10] alleviates the weaknesses of "low-fi" CRC sessions by aiding the process with computer tools.

Formal modeling, instead, is expressed using diagrammatic visual notations such as UML, the unified modeling language [11]. Tools such as ArgoUML or UML Lab[1] implement these methodologies. Although they support knowledge sharing and reduce maintenance problems, we believe they inhibit the creative nature of modeling: One spends more energy in reproducing the right steps for creating correct diagrams rather than in exploring possible design alternatives. Despite these drawbacks, UML editors are widely adopted and can be considered mainstream modeling tools.They have been also ported to new platforms, such as touch-based tablet computers, in the form of apps like ASTAH* UML PAD[2] and DRAWUML[3].

The emergence of touch-based tablet computers (*e.g.,* the iPad[4]) creates a middle ground that can combine the best of both formal and informal modeling: Touch-based devices stimulate a playful environment similar to pen & paper/whiteboard, their portability enables modeling in any setting and environment, and their computational power supports processing, storage, and maintenance of the produced models. These new technologies have been exploited also in the context of programming: YINYANG [12] is a visual programming language explicitly designed to allow software development on tablet computers. TOUCHDEVELOP [13], instead, introduces a simplified scripting language for programming directly on WindowsPhone smartphones. COFFEE TABLE [14] is an IDE built around a shared interactive desk.

Instead of using tablet computers as mobile whiteboards, or porting existing UML editors to touch-based devices, we devised a novel modeling approach, based a on a minimal set of elements, which we integrated into an iPad application called CEL. We took inspiration from previous research done in the context of novel IDEs ([15], [16] and [17]) and used abstraction to present a concise view of the entities of a software model.

CEL, depicted in Figure 1, has been developed to allow touch- and gesture-based mobile software modeling. The basic set of visual expressions, which developers can use to create and manipulate models, can in turn include other elements and are visualized as interconnected named entities. The minimal modeling language defined in CEL enables the creation of language agnostic models.

Interactions are performed using gestures, keeping keyboard- and widget-based interactions to the minimum. The intensive use of gestures introduces a learning phase, after which users can rapidly interact with the models under construction, via a uniform and simple user interface.

As illustrated in Figure 1, the main view of the tool depicts the model as interconnected matrix cells. The elements are laid out automatically, and to mitigate the problem of the small screen size and of the limited amount of information that can be displayed, CEL supports semantic zooming. Users can act on the matrix, instantiating new entities or relationships and manipulating the existing ones. They can also create a selection to focus on a group of elements and act on it. The models produced with CEL can be then exported as skeleton source code in any language of choice.

CEL is freely available at `http://cel.inf.usi.ch`.

[1] http://argouml.tigris.org/ and http://www.uml-lab.com/
[2] http://astah.net/editions/pad
[3] http://itunes.apple.com/us/app/draw-uml-for-ipad/id428468147
[4] http://www.apple.com/ipad/

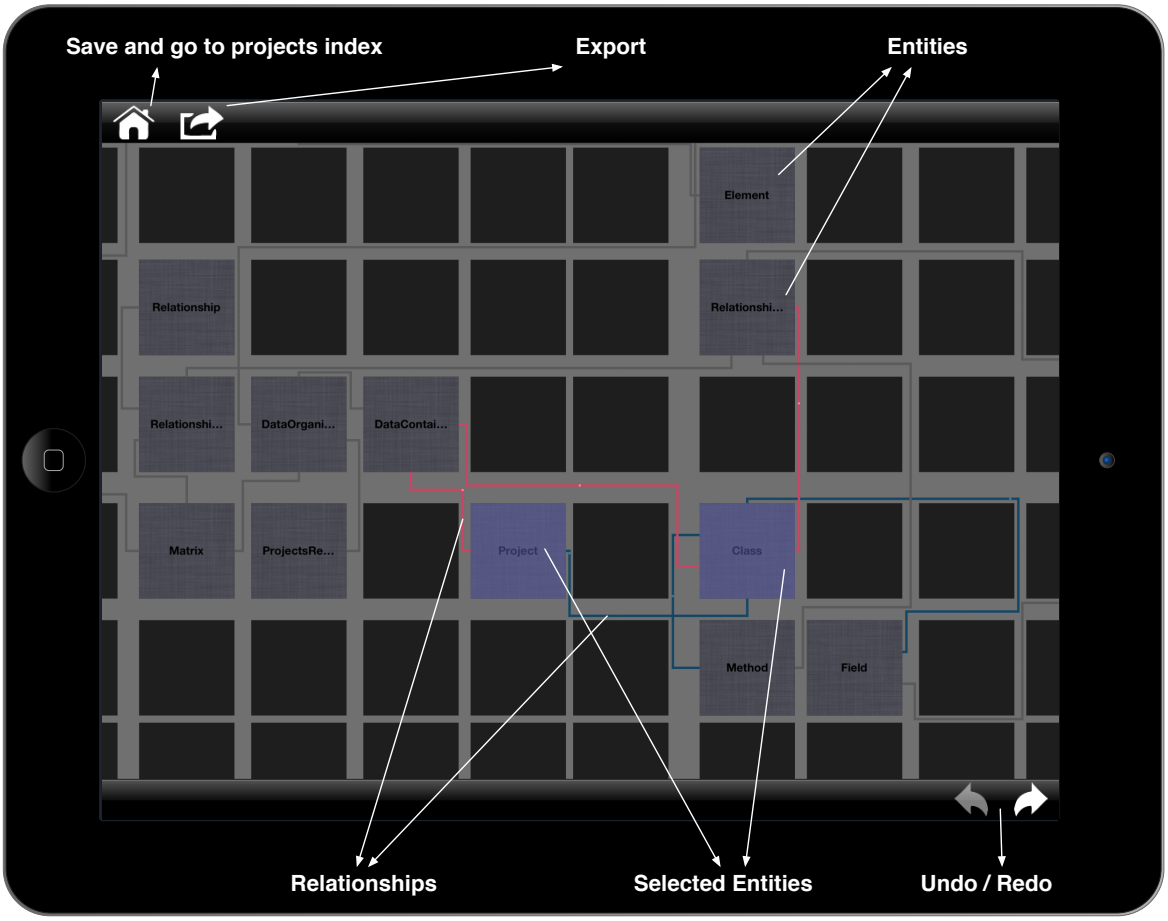


Fig. 1: Cel modeled in Cel on the iPad.

II. Cel in Detail

When designing and developing Cel we acted on four different levels to create a solid, powerful and usable tool that could best fit on a touch-based tablet computer as the iPad: the visual metaphor, the modeling philosophy, the user interface and the interaction methodologies.

A. The Visual Metaphor — Enter the Matrix

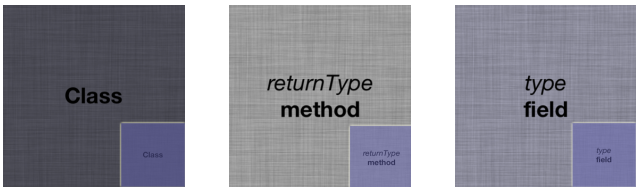
Cel has been built around a matrix metaphor, visible in Figure 1. The entities of a model are depicted as matrix cells, separated by a grid. This minimalistic interface introduces a constrained layout as opposed to free layouts used by UML editors. The matrix-based approach forces users to place entities inside the cells, yet it provides an intrinsic order and uniform visual appearance. The operations performed on one entity (*e.g.,* moving, deleting) do not affect the other ones (*e.g.,* there is no need for repositioning). Layout-related problems are solved automatically and users can concentrate on the modeling activity without having to deal with tedious details.

B. The Modeling Philosophy — Abundance of Simplicity

Our modeling philosophy considers only a restricted number of elements, because in the first stages of software system design one wants to deal only with few basic concepts.

Cel's main entities are classes, contained in the cells of the main matrix. Each class can contain behavioral entities (*i.e.,* methods) and state entities (*i.e.,* fields). The overall visual metaphor is consistent: Each class contains an inner matrix-based visualization in which the child elements are laid out.

Other types of entities (*e.g.,* interfaces, packages) overcomplicate software models and are not needed in the first stages of software design. We introduced an optional typing system to allow users to refine important entities when necessary. As depicted in Figure 2, the different entities are assigned a unique color, yet when selected, the blue overlay helps to exploit color similarity to make the selection appear as one single element.



(a) Class (b) Method (c) Field

Fig. 2: Entities in Cel. Bottom-right: selected entities.

Once the entities of a software system have been laid out, it is also fundamental to sketch the relationships among them. In CEL all relationships are directed and can either represent inheritance or a generic type of connection. Inheritance relationships are used to create a hierarchy and denote subtyping. Generic relationships are used to denote any other kind of connection (*i.e.,* use, containment, conceptual). We decided to limit the kinds of relationships because specific types of connections (*e.g.,* aggregation, composition) add unnecessary details to an initial model and, moreover, having more abstract connections leaves more freedom in the implementation phase. To lay out and display relationships, and to avoid edges crossing entities, we exploit the grid of the matrix (see Figure 3).

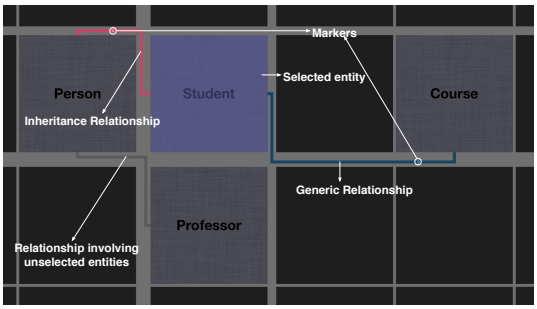
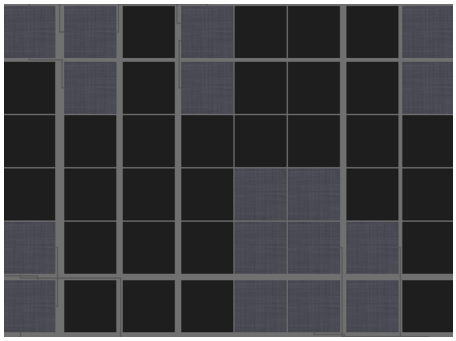


Fig. 3: CEL depicting relationships for a selected entity.

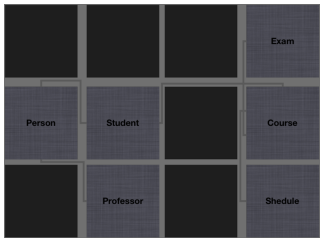
Although relationships are always visible (in a uniform light color), their details (*i.e.,* the type using a color and the direction using a moving marker) are shown only when at least one of the entities involved in the connection is selected. This way we show detailed information only for the connections involving entities already in focus, reducing information overload for the user. The paths of the relationships are computed using Dijkstra's algorithm and to keep intelligible paths we enlarge the channels of the grid when necessary. As a side-effect, users are able to spot entities involved in many relationships as they are surrounded by large channels. We tuned the strategy to calculate the paths in order to have a good trade-off between channel width and path length.

C. The User Interface — Zoom to Unveil

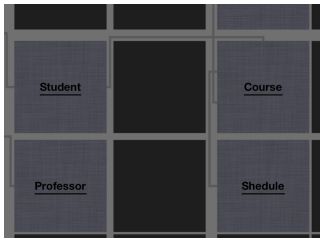
Applications for tablet computers have to deal with the small screen size and the limited amount of information that can be visualized. CEL solves these issues by treating the matrix as a *semantic zoomable interface*. When zooming in users progressively unveil details about the entities as well as new operations and actions. We opted for this solution because we argue that when users zoom out considerably they are interested in the "big picture" of the system, not in the details. On the contrary, when the number of displayed entities is small, the considerable amount of space occupied by each element can be exploited by adding more details. Moreover, allowing complex operations on small entities is risky, because the elements are difficult to distinguish and manipulate. This problem is automatically solved when zooming in. We introduced in CEL five different zoom levels, which are depicted in Figure 4.



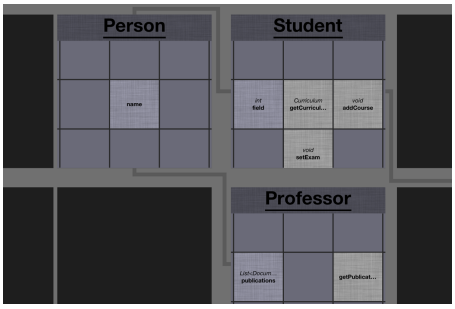
(a) Birds-Eye



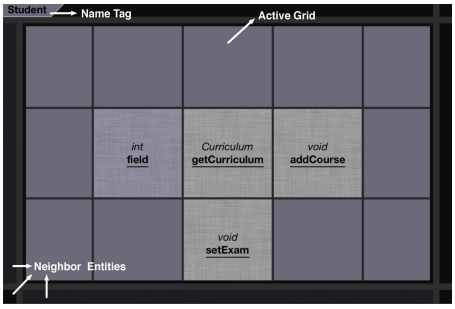
(b) Overview



(c) Edit



(d) Details



(e) Internals

Fig. 4: The five semantic levels supported by CEL.

(a) This level shows an overall view of the model, denying the editing of entities. (b) This level offers all the basic operations to modify the model and is used to operate on an entire region of the matrix. (c) This level offers elaborate interactions to modify the entities. (d) This level shows the composition of an entity and is used to rapidly inspect and navigate the internals. (e) This level is used to interact with the internals of an entity (the same five semantic levels are applied again).

D. The Interaction Methodologies — Gestures

Interaction was a main concern while designing Cel . Raskin stated that users should be able to focus only on the current task without any major interruption or disturbance [18]. To follow this guideline we avoid confirmation dialogues with a full undo/redo system that captures any relevant event, and we minimize waiting times by designing Cel to lock as few resources as possible and to defer time-consuming operations to separate threads.

Moreover, the interaction methodologies that users can employ to interact with the application are crucial. When dealing with touch-based tablet computers the absence of a pointing device and the difficulties created by the digitalized keyboard encourage the use of alternative methodologies. Gestures are movements/actions captured on a (multi-)touch sensing surface and are therefore perfectly suited to be the main interaction methodology to be used on a touch-based device. We decided to create also custom gestures. They do introduce a learning phase, but they also allow users to rapidly interact with the elements of the matrix. We limited as much as possible keyboard- and widget-based (toolbars with rarely used options) interaction. The gestures and the corresponding operations are summarized in Figure 5.

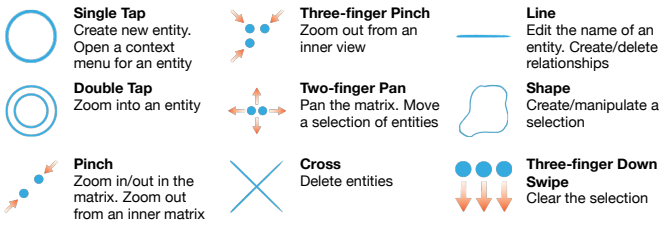


Fig. 5: Summary of the gestures used in Cel.

III. Conclusions

Modeling is a key activity in software engineering. We presented an innovative tool, Cel, for object-oriented software modeling. Our iPad application uses a custom matrix-based visual metaphor and a minimal set of constructs to support the modeling activity and to favor the rapid creation of software models and the exploration of different design alternatives. It exploits semantic zooming to address the problems created by the small screen size and to optimize the amount of displayed information. Cel mimics the freeform environment offered by lightweight modeling techniques by being a highly interactive, gesture-based portable device application. Our tool preserves the advantages of formal modeling methodologies by allowing the user to store, process, share and maintain the created software models. To bridge the gap between modeling and programming Cel allows users to export the produced models as skeleton source code in any language of choice [19].

We performed a controlled qualitative study to gather feedback on our approach and from this initial evaluation Cel seems at least as competitive as mainstream digital, UML-based, modeling approaches, which is very promising. The details about this study can be found in Lemma's thesis [19].

We believe there is a void middle ground between lightweight, informal –but volatile– modeling means, such as the whiteboard and paper drawings, and formal –but heavyweight– means, such as full-fledged UML editors. Our goal with Cel is to explore that middle ground.

Acknowledgment

We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "GSync" (SNF Project No. 129496).

References

[1] R. Lee and W. Tepfenhart, *Practical object-oriented development with UML and Java*, ser. An Alan R. Apt Book Series. Prentice Hall, 2002.
[2] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd ed. Prentice Hall, 2003.
[3] H. van Vliet, *Software Engineering - Principles and Practice*, 2nd ed. Wiley, 2000.
[4] J.-M. Favre, "Foundations of model (driven) (reverse) engineering: Models - episode i: Stories of the fidus papyrus and of the solarus," in *Language Engineering for Model-Driven Software Development*, 2004.
[5] D. Bellin and S. Simone, *The CRC Card Book*. Addison Wesley, 1997.
[6] D. Parnas, "Software aging," in *Proceedings 16th International Conference on Software Engineering*. IEEE Computer Society Press, 1994, pp. 279–287.
[7] G. C. Murphy, D. Notkin, and K. J. Sullivan, "Software reflexion models: Bridging the gap between design and implementation," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 364–380, 2001.
[8] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2007, pp. 557–566.
[9] N. Mangano, A. Baker, M. Dempsey, E. Navarro, and A. van der Hoek, "Software design sketching with calico," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 2010, pp. 23–32.
[10] F. Bott, S. Diehl, and R. Lutz, "Creww: collaborative requirements engineering with wii-remotes (nier track)," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 852–855.
[11] M. Fowler and K. Scott, *UML distilled - a brief guide to the Standard Object Modeling Language (2. ed.)*. Addison-Wesley-Longman, 2000.
[12] S. McDirmid, "Coding at the speed of touch," in *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*. ACM, 2011, pp. 61–76.
[13] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich, "Touchdevelop: programming cloud-connected mobile devices via touchscreen," in *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*. ACM, 2011, pp. 49–60.
[14] J. Hardy, C. Bull, G. Kotonya, and J. Whittle, "Digitally annexing desk space for software development (nier track)," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 812–815.
[15] F. Olivero, M. Lanza, and M. Lungu, "Gaucho: From integrated development environments to direct manipulation environments," in *Proceedings of FlexiTools 2010 (1st International Workshop on Flexible Modeling Tools)*, 2010.
[16] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. L. Jr., "Code bubbles: a working set-based interface for code understanding and maintenance," in *Proceedings of the 28th international conference on Human factors in computing systems*. ACM, 2010, pp. 2503–2512.
[17] R. DeLine and K. Rowan, "Code canvas: zooming towards better development environments," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*. ACM, 2010, pp. 207–210.
[18] J. Raskin, *The Humane Interface - New Directions for Designing Interactive Systems*. Addison-Wesley, 2000.
[19] R. Lemma, "Software modeling in essence," Master Thesis, Faculty of Informatics, University of Lugano, Jun. 2012.