Personal Programming for All

Trygve Reenskaug
Dept of Informatics, University of Oslo[1]

Abstract

Computer programming celebrated its platinum jubilee on the 21st of June, 2018. Exactly 70 years ago, the world's first programmer wrote the world's first program and then stored and executed it in the world's first stored program computer; affectionately known as *Baby*. The solitary Baby is morphing into billions of computers that are connected to become *a single, global machine*. Baby's control panel has morphed into graphical user interfaces (GUI) that empower everybody to augment their intellect. The consequences are deeply radical for individuals and society alike.

A new programming environment that I call BabyIDE, targets the single, global machine. Different GUIs support programmers having different mental models depending on their interests and proficiency. An MVC system architecture makes the program fade into the background and lets the user concentrate on satisfying his or her immediate needs. My approach is experimental. Smalltalk's universe of objects imitates the single, global machine and is my proving ground for various versions of BabyIDE. A great deal of work remains to make Personal Programming available to all and I am searching for a trailblazer who will take charge of it and take it out into the world.

Keywords:

Personal Programming
Novice Programming
Single, Global Machine
IOT
Smart Home
MVC
DCI
BabyIDE
Smalltalk
Object Orientation

[1] http://folk.uio.no/trygver/themes/pp/EllenVideo.mp4

1 Introduction

This is an interim report from my Personal Programming project. The final report and a video demonstrating the user's programming interface are in preparation.

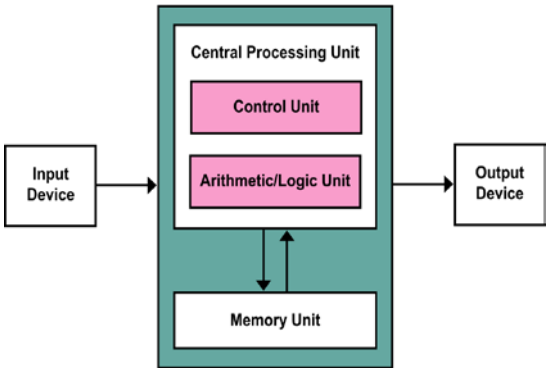
Professional Programmers create programs for many users in many places and situations. In contrast, *Personal Programmers* create programs for their own use here and now. Where the professional programmer must provide options for different user requirements at different times, the Personal Programmer simply creates code on the fly to cater for current needs. The Professional Programmer is a highly trained expert while the Personal Programmer has other interests and values simplicity and convenience over sophistication in programming.

We are entering the connected society. Every thing, every person will be connected through a global communication network. At the 2007 EG conference, Kevin Kelly shared a fun statistics: "The World Wide Web, as we know it, is only 5,000 days old". After having described the past, Kelly asks:

> "… what's going to happen in the next 5,000 days? So, I have a kind of a simple story, and it suggests that what we want to think about is this thing that we're making, this thing that has happened in 5,000 days -- that's all these computers, all these handhelds, all these cell phones, all these laptops, all these servers -- basically what we're getting out of all these connections is we're getting one machine. If there is only one machine, and our little handhelds and devices are actually just little windows into those machines, but that we're basically constructing a single, global machine." (Kelly, 2008)

I anticipate this single, global machine and choose to model it as collection of communicating objects. My concern is to empower Personal Programmers to master it. The first challenge is to create a mental model of computing that will form the foundation of user intuition. I look at two candidates: The von Neumann model and the DCI model.

Figure 1: A von Neumann model is a closed system.
(Wikipedia)

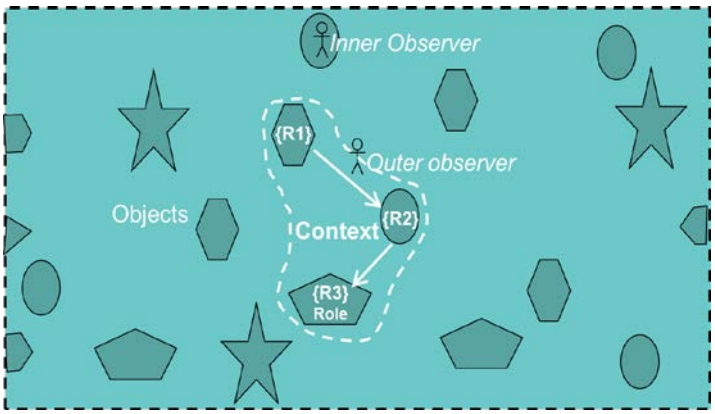


The *von Neumann model* is a closed system that can only interact with its environment through its input/output channels. Programming in the 1940s and 50s targeted this model, as do most mainstream programming languages today. I reject this model because its scope does not include the collaborating computers that billions of users will experience in the single, global machine.

The *DCI program model* is different in that it describes how an ensemble of objects collaborates to achieve a common goal. (Figure 2) illustrates the single, global machine where individual machines are represented by objects of different kinds. An object encapsulates state and behavior and can only interact with other objects through its message interface. An observer inside an object can only see how it is realized, e.g., as a class or a HTML script. An

observer in the space between the objects can only see the messages that flow when the objects collaborate within a Context. The program that drives this collaboration is distributed among the participating objects; each object being responsible for a part of the overall behavior.

Figure 2: The single, global computer



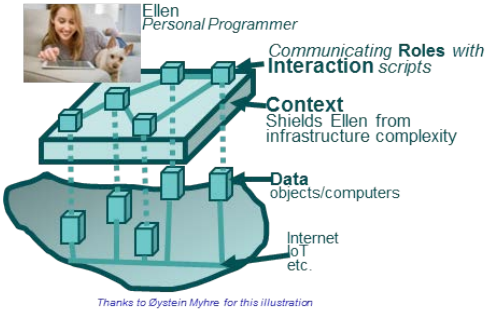
BabyIDE is a programming environment that leverages the power of MVC to make programming concrete and accessible to its users, the Personal Programmers. The BabyIDE *Model* is the distributed program as it exists in the participating objects. BabyIDE *Views* make parts of selected program objects visible and tangible according to the role they play when they collaborate. There can be different Views designed for different programmers ranging from the novice to the expert and from the domain-specific programmer to the general. (Reenskaug, 1978). The power and simplicity of programming with BabyIDE will be demonstrated with a video example (section 2).

The DCI programming paradigm serves as the Personal Programmer's mental model of programming. The DCI ***D****ata* is the universe of reachable objects. These objects may be predefined, created automatically, or created explicitly by the programmer. The DCI ***C****ontext* encapsulates the objects that are selected to play *Roles* in a DCI ***I****nteraction* at runtime. (http://fullOO.info)

A theatre analogy is often used to explain DCI. The *DCI Data* objects are like actors; they may be working or "resting". The *DCI Context* is like a stage where actors perform their roles. The DCI Interaction is like the text of the actor's part. Oxford English Dictionary *role*: "Early 17th century: from French rôle, from obsolete French roule ‘roll’, referring originally to the roll of paper on which the actor's part was written". In DCI, the roll of paper is called a role script.

The DCI Context forms an abstraction barrier that shields the Personal Programmer from the complexities of the Data objects and their infrastructure (Figure 3).

Figure 3: The DCI Context as a barrier between programmer and the distributed program objects.



The rest of the article is organized as follows:

Section 2: *Ellen's Smart Alarm Clock.* is the narration for a planned video showing how Ellen programs a smart alarm clock.

Section 3: Conclusion.

Section 4: Acknowledgements.

Section 5: References.

2 Ellen's Smart Alarm Clock

This is the narration script for demonstrating of how Ellen uses BabyIDE to program a smart alarm clock.

Figure 4: Ellen's Internet of Things

We live in amazing times. The things around us come alive and serve us as long as we learn how to control them -- We need to learn how to program.

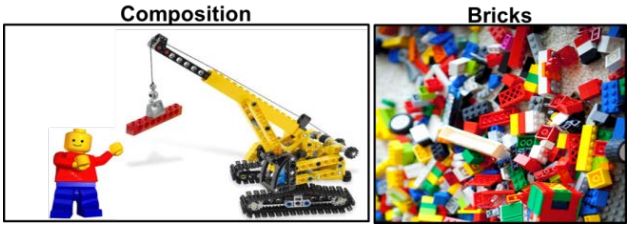
Figure 5: Ellen is one of many Personal Programmers

Let's single out Ellen, as a person among many. She wants to learn how to control her things, and she challenges us with her first, simple example:

Figure 6: Ellen hiking

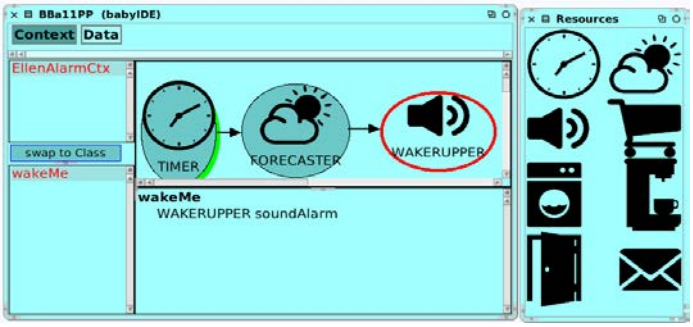
Ellen plans to go on a long hike on the morrow, but only if it's going to be a dry day. So, she needs to program an alarm clock that checks the weather forecast before it wakes her.

Figure 7: A Lego construction toy



Ellen is fully at home with the idea of composition. She has, for example, experience with her Lego construction toy. It provides her with a bucketful of bricks of different shapes and colors. She picks bricks one by one and joins them to compose whatever she is building.

Figure 8: Ellen's IDE.



A programming paradigm, that I have called DCI, lets Ellen compose her program in the same way. The Lego bricks become objects that that represent things in Ellen's world. She selects one by one and connects them in a Context to become her program.

One of the objects represents a bedside alarm that has replaced her old alarm clock. She picks it up, moves into her Context, and names it: WAKERUPPER.

While a Lego brick is a dead chunk of plastic, Ellen's objects are smart, they can do anything a computer can do. So she augments the WAKERUPPER with a script that tells it what she wants it to do for her:

```
wakeMe
    WAKERUPPER soundAlarm.
```

She tests it, and it works. (The alarm sounds)

Next, she needs a weather service, finds it in her box of objects, drags it in, connects it up, and tells it what she wants it to do for her:

```
checkWeather
    FORECASTER expectedRainfall = 0
            ifTrue: [WAKERUPPER wakeMe]
```

And then, finally, she selects a TIMER that will perform this program tomorrow morning:

```
waitTillMorning
    TIMER waitUntil: '06:00'.
    FORECASTER checkWeather.
```

That's it, the program is complete and Ellen can set her alarm clock.

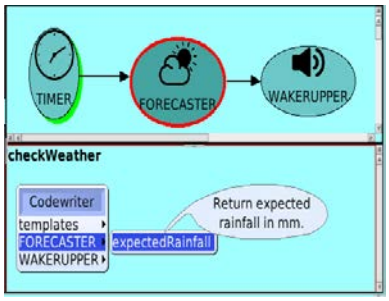
Comment:

The idea of role scripts is new to Ellen, and we endeavor to introduce it as smoothly as possible. We have chosen Squeak/Smalltalk for her scripting language because it is conceptually simple and is easy to read for the novice. Ellen creates and edits a role script in a text editor (middle-bottom pane in **Figure 8**). This is a conventional *"remember and type"* kind of user interface; it requires that the language is part of her active vocabulary. For example, she must remember the language to be able to write:

```
checkWeather
    FORECASTER expectedRainfall = 0
            ifTrue: [WAKERUPPER wakeMe]
```

We prefer to give her a *"see and select"* kind of interface so that only requires her to read the code; it can be part of her passive vocabulary. The exact nature of this interface is for further study. We have created a preliminary solution in the form of a 2-level menu of code templates (Figure 9). Balloon texts help Ellen when the mouse hovers over a menu item.

Figure 9: Coding by menu selection

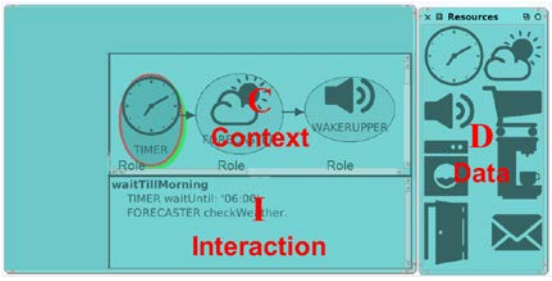


A benefit of this form of visual programming is that its generated code is visible and editable, thus helping Ellen to gradually include Personal Programming in her active vocabulary.

Figure 10: Ellen's Mental Model of computing

You may have noticed that Ellen's code is far from conventional. Ellen has an object-oriented mental model of what goes on. More specifically, her model is based on a programming paradigm called DCI.

Figure 11Ellen's DCI window for Personal Programming

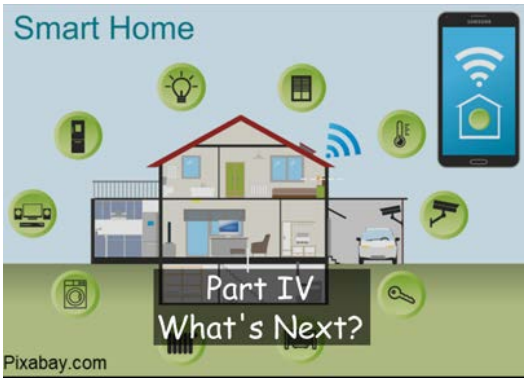


This paradigm gives her a powerful separation of concerns in three orthogonal projections. They are called *Data*, *Context*, and *Interaction*.

- The **Data** are stand-alone objects such as the objects that represent things in Ellen's world and services available on the Web.
- The **Context**s are structures of roles that selected objects play to meet Ellen's goals.
- And finally, an **Interaction** is the code that augments the objects to make things happen.

DCI scales, so that Ellen can grow from the novice Personal Programmer to the expert without ever having to change her fundamental mental model of computing.

Figure 12: What's next?



The programming tool that I have shown here is an experimental proof-of-concept implementation that I have called BabyIDE. Much work is needed to develop it into a general tool that can be used by all.

Figure 13: More at http://folk.uio.no/trygver/

I'm an octogenarian and I lack the necessary energy for this work. So I'm searching for an innovator who will identify with the goal and realize its potential.

The alarm sounds, the world is waiting, and it's time to wake up. (Alarm sounds)

3 Conclusion

There are many initiatives for teaching programming to non-professionals, particularly to kids. They are all targeted at the von Neumann model. I rejected these solutions in the introduction and deem them outside the scope of this article.

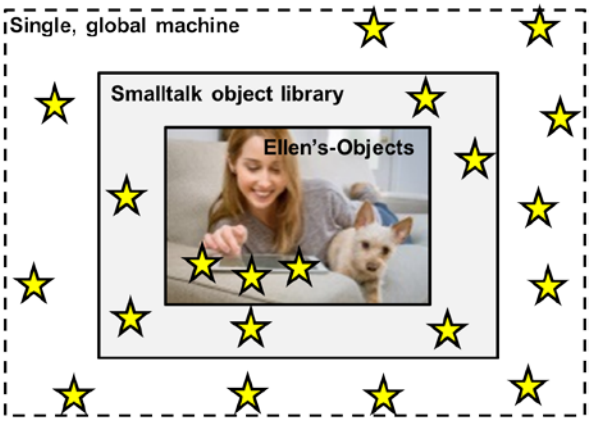
I set out to empower Ellen, a generic Personal Programmer, to harness the resources of her personal devices as well as the resources of the emerging single, global machine. The first challenge was to offer her a mental model of her digital environment. I chose a model consisting of objects and objects only: *"An object is an entity that encapsulates state and behavior that can only be accessed through the object's message interface.*[5]*"* The second challenge was to base her IDE on a concept that she will experience as being simple and

[5] Alan Kay coined the term object-orientation: *"Thus its semantics are a bit like having thousands and thousands of computers all hooked together by a very fast network"* (Kay, 1993). Note that the attention is on the runtime, compile-time ideas such as classes are of secondary importance.

intuitive. I selected the idea of *composition*, an idea that Ellen learned early in life and that forms part of her intuition. It is very likely that she has experience with Lego, the toy where children compose structures from bricks and bigger structures from sub-structures recursively. There is a short leap to see the objects as the Lego bricks of computing and structures of communicating objects as structures of Lego bricks.

From Ellen's point of view, all objects are similar and are handled in the same way. They are shown as yellow starts in (Figure 14). Under the hood, they come from different sources and have different access mechanisms: The objects of the single, global machine are accessed over the Web. Ellen's IDE is implemented in Smalltalk. For security reasons, Ellen cannot be allowed to modify the Smalltalk class library, but she can instantiate its classes. Ellen can freely create and instantiate her personal classes. Mechanisms in BabyIDE maintains the illusion that all the objects appear as the same kind of entity.

Figure 14: Ellen's object computer.
ObjectComputer-1.png



The DCI programming paradigm was launched in 2008 and has an active community centered around the object-composition@googlegroups.com mailing list and its home page: http://fullOO.info. The community has adapted DCI to a number of programming languages. Contact the mailig list for details. DCI has been compared to Java in a controlled experiment that showed that DCI code is more readable than Java code (Valdecantos, 2016).

Personal Programming with its modified BabyIDE has been my solitary project financed with my pension. The IDE is a demonstration program and is not ready for general use. The next step should be to establish a project with its own financing and team of researchers.

Personal Programming has been demonstrated to a number of visitors. The few programmers among the visitors didn't see the point; they neither needed DCI nor Personal Programming, thank you. All the non-programmers appeared to see the point and all had ideas about services they would like to develop for their own use. One of them, a farmer, had a number of useful applications. One was to monitor her beef calves. There was probably something wrong with a calf that was off its food and she should give it extra attention. Another was to monitor her cattle food silos; her program would alert her when the silos were nearly empty and help her select and order the next lot of grain feed. The farmer had been using computers for many years but had never written a program. She came with a pertinent observation: *"This doesn't feel like programming at all."* She was right, of course. Ellen's composition of her smart clock was more like setting up a playlist in a music server than programming in the conventional sense of the word. There was no source file, no compilation stage, and no concrete program. Instead, Ellen selected objects from her environment and applied the DCI paradigm to control the flow of messages that reifies her needs.

The response from the few non-programmer visitors was encouraging. I do not believe they would have been as understanding and creative if I had demonstrated how to write a simple Python program for the same purpose. I claim that the programming technology presented in this article will lead to programs that are easy to read, write, and grok for the Personal Programmer. Personal Programming has a low entry threshold, yet Ellen can learn more and do more until she can do everything that the single, global computer can do.

4 Acknowledgements

The work that has led to the DCI paradigm, BabyIDE, and Personal Programming has taken many years of ups and downs. I could not have stayed the distance if had not been for the encouragement I received from men I deeply respect, the foremost being Dave Thomas and Bran Selic.

I have long known James O. Coplien (Cope) and thank him for many rewarding discussions over the years. Our common ground has been our focus on people. The value of a system is its value for its users. Users can be the end users of an application or its developers using a programming environment. We had both been following our separate paths when searching for a common truth we both have felt must be out there somewhere. On Aug 28, 2008, I launched a new programming paradigm I called DCI - Data, Context, and Interaction. It was accompanied by BabyIDE, an Interactive Development Environment written in Squeak[6]. I spread the good news to a large number of people over the Web. There was no response except one: Cope wrote: *"Til lykke, Trygve. It's not often that one can claim two great life accomplishments in one life."* At long last we joined forces to further the DCI ideas. I am grateful for Cope's invaluable contributions to this work. Also for the dissemination of DCI which could not have happened without him.

My sincere thanks to the active community centered around the object-composition@googlegroups.com mailing list for their contributions to the DCI paradigm and its dissemination.

The concepts of Personal Programming and Ellen's object computer stem in part from Smalltalk that was created at Xerox PARC by Alan Kay, Dan Ingalls, Adele Goldberg and the Learning Research Group .The value of their disruptive ideas cannot be overestimated.

5 Bibliography

Alan Kay, A. G., 1977. Personal Dynamic Media. *Computer*, Issue March 1977, pp. 31-41.

Alaya, M. B., 2015. *Towards interoperability, self-management, and scalability for machine-to-machine systems. Networking and Internet Architecture [cs,* s.l.: Universite Toulouse III Paul Sabatier,.

Choi, J. K. & Kim, H. J., 2016. *Analysis of Digital Data Technologies Toward Future Data Eco-Society,* s.l.: International Telecommunication Union.

---

[6] BabyIDE works under Squeak version 3.10.2. It is not easily converted to later, more complicated versions.

Ducasse, S., 2005. *Squeak: Learn Programming with Robots.* s.l.:Apress.

Emery, F. & Thorsrud, E., 1976. *Democracy at Work: The Report of the Norwegian Industrial Democracy Program (International Series on the Quality of Working Life).* s.l.:s.n.

Galas, C. & Freudenberg, R., 2010. *Learning with Squeak Etoys,* s.l.: s.n.

Kay, A., 1972. A Personal Computer for Children of All Ages. *ACM '72 Proceedings of the ACM annual conference - Volume 1*, 1 August, p. Article No 1.

Kay, A., 1989. *User Interface: A Personal View.* [Online]
Available at: http://www.vpri.org/pdf/hc_user_interface.pdf

Kay, A., 1993. The Early History of Smalltalk. *ACM SIGPLAN Notices archive*, March, pp. 69-95.

Kelly, K., 2008. *The next 5,000 days of the web.* [Online]
Available at:
https://www.ted.com/talks/kevin_kelly_on_the_next_5_000_days_of_the_web#t-8111

Reenskaug, T., 1973. *Administrative control in the shipyard.* Tokyo, ICCAS conference. Also at http://heim.ifi.uio.no/~trygver/1973/iccas/1973-08-ICCAS.pdf.

Reenskaug, T., 1977. Prokon/Plan - a Modelling Tool for project Planning and Control. *IFIP Congress*, pp. 717--721.

Reenskaug, T., 1978. *MVC, XeroxX PARC 1978-79.* [Online]
Available at: http://folk.uio.no/trygver/themes/mvc/mvc-index.html

Reenskaug, T., 1996. *Working With Objects. The OORAM Software Engineering Method.*. Greenwich: Manning.

Valdecantos, H. A., 2016. *An empirical study on code comprehension: DCI compared to OO.* [Online]
Available at: http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=10401&context=theses

Wiegers, K. E., 2002. *Seven Truths About Peer Reviews.* [Online]
Available at: http://www.processimpact.com/articles/seven_truths.html

Wirfs-Brock, R. & McKean, A., 2003. *Object Design. Roles, Responsibilities, and Collaborations.*. Boston, MA: Addison-Wesley.