

Cuis Smalltalk

The Smalltalk with a fully zoomable, vector graphics GUI

<http://www.cuis-smalltalk.org>

<https://github.com/Cuis-Smalltalk/Cuis-Smalltalk-Dev>

Juan Vuletich, November 2022

Maybe some of you know me. I'm Juan Vuletich. I develop Cuis Smalltalk.

I want to start by saying 'Thank You' to some of the people that made all this possible. By "all this" I mean Smalltalk, me learning about Smalltalk, 25 years of professional work on Smalltalk, and the active and vibrant community we have today, especially in Argentina. Thank you Alan, Dan and Adele. Thank you Máximo and Leandro. Thank you Hernán. Thank you to my classmates and colleagues, and to the companies I have worked and work for. Thanks to all the Cuis and Smalltalk community.

Some people here have heard me talk about Cuis before, or may be even used it. Others know little about it. In this talk I'll tell you why does Cuis Smalltalk exist, which problems it tries to solve, and why this moment, November 2022, is the most important time in the history of the project.

Yeah, really!

I will tell you about some things that many of you have heard me talk about before. Most of those things were then in development. Today, all those things are part of Cuis Smalltalk, so that's the big news here.

Cuis is not just "let's do a new Smalltalk and see where it goes". Cuis was started to attack several important problems I saw in Squeak and the other Smalltalk systems, even knowing it would take several years to solve them.

The Smalltalk-80 spirit

“Design Principles Behind Smalltalk” by Dan Ingalls

- Personal Mastery: *Must be simple enough*
- Good Design: *Use few, general parts*
- Modularity and Encapsulation: *Inner details not visible from outside*
- Factoring: *Do not repeat yourself*
- System Boundaries: *Encapsulation of the supporting platform*
- Reactive Principle: *Visible objects should be interactive*

Some words about what happened before Cuis.

Smalltalk-80 was developed following a set of ideas described in “Design Principles Behind Smalltalk”. This article, published in the August 1981 issue of the Byte magazine, was the first description of the Smalltalk system that would reach a wide audience.

Here, Dan Ingalls tells us that Smalltalk-80 was built to provide computer support for the creative spirit in everyone.

The objective of these specific ideas, these are *specific* ideas, is to allow programmers to understand, modify and extend any part of the system. The usual separation between our work and a tool that was provided by a third party is completely removed.

Removing unneeded barriers gives developers an unprecedented level of freedom.

Back to the Future: The Squeak Project

1996 - 1998 A modernized, Open Source, Smalltalk-80

- By the same people who did Smalltalk-80 at Xerox
- 32 bit VM and Object Memory
- 8, 16 and 32 bit per pixel Color
- Morphic graphics and UI framework
- Sound and Music Synthesis

1999 - 2009 A platform for Education

- Etoys
- Scratch

In 1996 Squeak Smalltalk was born. To develop Squeak, Alan Kay formed a dream team that included Dan Ingalls, Ted Kehler and Scott Wallace from the original Smalltalk group, together with John Maloney from the team that built Self and Morphic. Later other talented and creative people joined the team.

Squeak was a direct descendant of Smalltalk-80, and it was true to its goals and design ideas. Making it open source and freely available on the Web meant that an enthusiast community of collaborators grew around it.

Squeak later made its focus to become a platform for education, around 1999 perhaps, drifting away a bit from the design principles that had guided Smalltalk-80. This started a whole new generation of educative software, that includes Etoys and Scratch, and other tile-based programming systems for education.

Smalltalk-80 and Squeak

Unique implementation strategies

- The UI framework is fully written in Smalltalk.
- The BitBlt graphics engine is minimal. In Squeak it is written in Smalltalk.
- Text layout and text editors are written in Smalltalk.
- StrikeFonts are drawn by Smalltalk, using only BitBlt.
- In Squeak, the VM is written in Smalltalk.

Smalltalk-80 and Squeak are different from commercial Smalltalks in many ways. Most have to do with portability and control, and are consequences of applying or not, the Design Principles.

The features offered by the Virtual Machine to the Smalltalk environment are minimalistic and platform agnostic. More important, they don't include arbitrary decisions on how the system, and especially User Interfaces build with it, should work.

This means that GUIs as different as MVC in Smalltalk-80 and Morphic in Squeak use exactly the same VM services. The level of flexibility offered to the developer is undeniable.

But most Smalltalk systems abandoned this approach, becoming more closely integrated with some host Operating System. When Squeak was started from Apple Smalltalk, one of the things that needed to be done, and I'm quoting the 'Back to the Future' paper was to "eliminate uses of Mac Toolbox calls to restore Smalltalk-80 portability".

This should tell us something, right?

The Problems

Squeak gave us a free Smalltalk-80, but a few years later:

- Squeak can't evolve any more
- Morphic is tied to Display pixels
- Available graphics primitives are limited
- Morphic programming is too difficult
- Text has low visual quality
- Unicode support is complex and error prone

But a few years later Squeak was left in the hands of its community. Those using Squeak became aware of some problems.

- Squeak can't evolve easily any more because of to excess complexity.
- Morphic is tied to Display pixels. Attempts to solve this limitation (first using WarpBlit and later the Balloon engine by Andreas Raab) were not enough.
- The graphics engine should provide services to draw curves and other graphics primitives, not just lines, boxes and text
- Writing new Morphs is too hard. Programmers must deal all the geometry and drawing themselves. Morphic is complex, and it doesn't try hard enough to ease the life developers.
- StrikeFonts were OK in the 70s and 80s, but 1 bit per pixel bitmap fonts don't look good on modern LCD screens anymore.
- The Unicode support later added to Squeak requires programmers to have deep knowledge of text encodings to avoid displaying garbage.

I understood that the root cause of the problems Squeak was running into, was leaving behind the Design Principles. I decided to solve these problems one by one, by following them again.

The birth of Cuis Smalltalk

2003 - Goals for a new Smalltalk project

- Independence of Pixel Density
- Zoomable User Interfaces
- Vector Graphics based UIs
- A better way to do Unicode

Me: "Should I wait?"

Alan Kay: "Good question. The answer is always the same: No. You should never wait."



6

There weren't any retina displays yet, but it was clear to me that GUIs designed for a specific pixel density had no future. I wanted a Morphic system where everything was specified independently of pixel size or resolution.

But a large community of people, now without a strong leadership as we had before, and with a deep devotion for their heritage, was not ready for the disruptive kind changes that were needed to go where I wanted to go.

In 2003 I got the chance to meet with Alan Kay. During that talk, I told him about these ideas. About that time, Andreas Raab was working on his own replacement for Morphic, he called Tweak. So I asked Alan: "Should I wait for Tweak, and maybe adopt it?". He said: "That is a very good question. And the answer is always the same. No, you should not wait. You should never wait."

I followed his advice. Cuis Smalltalk was derived from Squeak in 2004, and ever since it has been evolving into a modern incarnation of the Smalltalk-80 spirit.

Some project landmarks

Steady progress over a long time

- 2004 - Work on cleanup and reduction of Morphic begins
- 2006 - Work on the new Morphic and a new Vector graphics engine begins
- 2007 - First presentation and demo at Smalltalks 2007 (Buenos Aires)
- 2008 - New Vector Graphics rasterization algorithm (disclosed in 2013)
- 2009 - Cuis 1.0 published, Cuis as a separate Open Source project is started
- 2010 - Support for true Closures, added to the OpenSmalltalk VM
- 2012 - Support for loadable Code Packages
- 2013 - "Prefiltering Antialiasing for General Vector Graphics" published
- 2016 - 64 and 32 bit support using the Cog Spur VM from OpenSmalltalk
- 2019 - TrueType support, better quality than FreeType, ClearType or Quartz
- 2020 - Integration of VectorGraphics, with SVG support in official Cuis
- 2021 - Integration of VectorGraphicsEngine high performance plugin in OpenSmalltalk official builds
- 2022 - Unicode support added, "Unicode support in Cuis Smalltalk" published

7

These are some project landmarks over 19 years.

During this time, Cuis was updated to the latest developments in the Squeak world, adopting the OpenSmalltalk VM, that included support for true block closures, a high performance jitter, and later, 64 bit support.

Other developments were done specifically for Cuis. This includes a huge cleanup, reduction and refactor of almost every part of the system. Additionally, I added a source code format for optional code packages, and a workflow to enable collaboration on GitHub.

But the most important pieces of work were done to solve the problems I found in Squeak. Let's go over them.

Problem: Global Pixel Coordinates

- Global pixel coordinates are everywhere!
- All submorphs at any level live in the same global space
- Morphic programming becomes too complex

Solution: Morphic 3, a deep framework redesign

- Floating Point numbers for coordinates
- Arbitrary linear geometry transformations
- Local Coordinates to the Morph's space

Global Coordinates.

Because every Morph uses global integer pixel coordinates, all the GUI related code, needs to deal with them all the time. Anybody who tried to build new Morphs know how complicated things can get. Just moving a morph around on the screen will mean updating the coordinates for dozens of nested submorphs. Scaling and rotating become very complicated.

The solution was already suggested by John Maloney in an interview done for the SqueakNews electronic magazine in 2001. Every Morph should define the space and coordinate system in which its submorphs will live. Coordinates are real numbers, not integers. The programmer of a Morph only needs to deal with the geometry that makes for that morph, and don't care about anything else. All conversions and other technical details should be handled by the framework.

Converting Morphic from the old design into this new design took several years and was done in small, incremental steps. Assuming that it is reasonable to call "Morphic 1" the original implementation in Self, and "Morphic 2" the version in Squeak, I decided that "Morphic 3" was a good name for this new design.

Problem: Quality of graphics

- Quality of graphics not good (unless drawn by graphics designers)
- Pixel Coverage anti-aliasing still creates pixellation
- Set of available drawing primitives too limited

Solution: A new approach to rasterization of graphics

- Apply Signal Processing theory to Anti Aliasing
- “Prefiltering Antialiasing for General Vector Graphics” published in 2013
- A new Vector Graphics Engine, API inspired by SVG
- Theoretically correct, sub pixel sampled graphics and text now possible

Visual quality of Graphics.

In those times, we're talking 20 years ago, the quality of graphics drawn by most applications was not very good, and LCD displays only made this more evident. Commercial applications tried to make this less obvious with the heavy use of hand drawn bitmaps done by professional graphics designers. The reason is that, even if they try to do some Anti Aliasing, the conventional pixel coverage technique can't really avoid pixellation.

Most graphics software is still written with the idea of pixels as square tiles covering the screen.

In addition, the set of drawing primitives provided by Squeak is limited.

The solution involved some real work.

The theory to describe aliasing and how to avoid it has been part of the fields of Optics and Signal Processing for decades. Designers of photo and movie cameras know very well that they need to match the optical system (the lens) to the sensor, and usually add a low pass filter. Designers of digital sound recording systems know they must do a proper low pass filtering before sampling audio.

What I needed was a practical way to do the same thing for Vector Graphics. After maybe a couple of years of fighting this problem, the solution I found is so simple and effective, that I decided I needed to protect it with a Defensive Publication before publishing the code. I could go into more detail on why that is important, but maybe there's no need. I hope.

Problem: Quality of Text

- Quality of text in early Squeak not good
- Improved later, still complicated to add new fonts
- Text can't be rotated or scaled

Solution: Reuse the good stuff

- Use the new Vector Graphics Engine also for doing text!
- Build a Smalltalk model for TrueType glyphs using Floating Point Beziers
- High quality, subpixel sampled text using TrueType fonts
- Even applying arbitrary linear geometric transformations

Visual quality of Text.

The quality of text in Squeak was not very good, and LCD displays made this more evident.

Around that time frame, mainstream operating systems adopted text specific rasterization engines, and they developed special tricks to reduce blurriness and make the result look better, sometimes even getting patents on them. These text specific rasterizers include ClearType, FreeType and Quartz. The tricks they use, prevent the implementation of features like rotated text, and they result in incorrect geometry, and text that just looks weird. Compare, for example, Windows Notepad and Acrobat Reader, and maybe zoom the display and see how different their text looks.

But the Vector Graphics Engine I wrote for Cuis is geometrically correct and subpixel sampled. It can offer the same level of quality as text specific engines, but can also do arbitrarily rotated and scaled text. This makes text fit naturally into the improved Morphic system.

The solution also involved designing a new Smalltalk model for TrueType, using Floating Point coordinates to represent the Bezier curves that Glyphs are made of, and adding new services to Vector Graphics Engine to deal with Text and TrueType.

Problem: Unicode is complicated and error prone

- This is of course not only in Smalltalk.
- “Unicode support in Cuis Smalltalk”, to be published in FAST Workshop on Smalltalk Related Technologies

Why is Unicode complicated?

- Unicode is hard to use in any programming language
- Naive code breaks quickly when dealing with unexpected Unicode content
- Why?

Unicode.

This big problem is not limited to Smalltalk!

What follows is a short version the paper I wrote for the FAST Workshop that is part of this conference, named “Unicode support in Cuis Smalltalk”. I suggest that you take a look at the paper.

So, Why does everybody have trouble with Unicode? And by “everybody”, I mean EVERYBODY, ok?

The main reason is backwards compatibility. In Cuis, we were able to take a fresh look at Unicode and try to come up with good solutions, without being limited by compatibility with existing applications.

What is a String?

- A String is a sequence of bytes that represent characters

What is an ASCII String?

- An ASCII String is a sequence of bytes where each byte represent a character

What is an Unicode String?

- An Unicode String is a sequence of bytes where several bytes represent a single character

Let's ask these basic questions.

What is a String?

Everybody knows that a String is a sequence of bytes that represent characters. This is so, at least, since C, ok?

What is an ASCII String? Well, the same as a String, right? It is a sequence of bytes that represent characters.

So, what is an Unicode String? Well, it is a String, but now several bytes may represent a single character.

Right?

No. This is not right. This is wrong!

What is a String in a high level language like Smalltalk?

- String is a data type with an external protocol that behaves like a sequence of Characters
- Character is a data type that represents an element in a written language. Its external protocol reflects that.

Their internal representation is irrelevant to users and is protected by encapsulation

There may be a family of implementations that are interchangeable thanks to polymorphism

Most users (i.e. programmers) don't care about these details at all!

Well, this is Smalltalk, right?

A language that was designed to let any programmer define new data types.

We need to think a bit what we would like Strings and Characters to look like. What protocol makes more sense for them. And then, implement only that, as the public protocol for the related classes.

So, a String is a data type with an external protocol that behaves like a sequence of Characters. *Not what we said before.*

And a Character is a data type that represents an element in a written language, and its external protocol must reflect that.

Their internal representation is irrelevant to users and is protected by encapsulation. There may be a family of implementations that are interchangeable thanks to polymorphism.

And most users, most programmers, don't care about these details at all.

A String should never be

- A sequence of bytes whose meaning is externally determined

```
aString encode: anEncoding -> String  
aString decode: anEncoding -> String  
This is Bad. Wrong. Unacceptable.
```

- A sequence of bytes or words at all

```
#at: or #at:put: don't handle words or bytes. Only Characters.  
#byteAt: or #byteAt:put: should simply not exist.
```

Strings should be just like Numbers

- There are many kinds of them
- They use different kinds of bit patterns internally
- You don't usually care about that
- They can convert themselves as needed to be compatible in operations

The language and libraries should never require the programmer to know the encoding of some String, only because the implementation doesn't know by itself.

Encoding (from String to Bytes) should always answer a ByteArray. Decoding, to get a String, should only be done on ByteArrays.

An operation that takes a String, encodes it with some encoding, and the answer is a String is wrong. It is broken and the answer makes no sense.

Access methods should never handle words or bytes, just Characters. The only exception may be private methods, used only by the implementation itself.

In short, Strings should work just like Numbers!

- There are many kinds of them
- They use different kinds of bit patterns internally
- You don't usually care about that
- They can convert themselves into each other's types as needed to be compatible in operations

What else is needed for Unicode?

TrueType, Text Display, Text Editors and FileStreams

- Expanded the TrueType font model to handle the whole range of Unicode Code Points, with accessors that take their UTF-8 bytes.
- Expanded the Text Display services in VectorGraphicsEngine to handle UTF-8 encoded text, using the Unicode ready TrueType fonts.
- Made Cuis Text Editors handle Utf8Strings.
- Replaced StandardFileStream with a new class named UniFileStream.

Possible modes are #readBytes, #readByteCharacters, #readUtf8Strings, #readUtf32Strings.

I designed a new data structure to represent glyphs, this is, the contours built with Bezier curves. It uses only two objects per TrueType font: One large IntegerArray for indexing, and one even larger FloatArray for geometric parameters.

This data structure let's us find the position in the FloatArray for the information about any code point using only 1 to 4 array accesses using the already available UTF-8 bytes. It is integrated in the VM plugin for the Vector Engine, making text display very fast, even if applying any geometric transformation as Morphic 3 allows.

Making the Text Editors work on Unicode Strings was very simple, because of polymorphism between String classes. Little code needed modification, and it was only because Unicode Strings are immutable (they don't have the #at:put: operation).

To support Unicode text files, I decided to create a new File Stream class. The reason is that the old file modes "text" and "binary" no longer make sense. This new class can write Strings and Characters (in UTF-8), but also bytes and words. The "mode" is only used for reading, for how to interpret a file. The included modes are UTF8, the default, and also bytes or possibly other encodings for Strings.

What else is needed for Unicode?

Unicode in Smalltalk code

- Use of Unicode in comments and String literals is done
- Modified Smalltalk Parser to allow letters from any alphabet in names
- Also to allow Unicode mathematical symbols as binary operators

As everything, from files to text editors, is already Unicode ready, nothing else is needed to use Unicode in code comments and String literals.

But the Smalltalk language lets us only use letters in selectors, class names and variable names. As Unicode “knows” which code points are letters in some script, all that was needed was to modify Parser to take advantage of this information.

Now Smalltalk models can be written using the alphabet of the language for which the model is intended. Not everybody will use this, but it can be helpful for software that is very specific to some country. To make the entities in a Smalltalk model better reflect reality, you can write the names that are used in that language using its own alphabet.

In the same way, I made Parser accept any Unicode mathematical symbols as binary operators, not just the ASCII ones. This means a lot more expressive power for Math related code.

The evolution of Cuis meant developing

- A significant evolution of the Morphic GUI framework
- A new graphics engine, doing Vector Graphics, and based on Signal Processing theory.
- Modernized Text Layout and Text Editors.
- TrueType fonts, modeled in Smalltalk.
- Unicode support, done in the true Smalltalk way.

17

All this required a redesigned, more advanced User Interface framework, that besides being more flexible, it is more comfortable and easier to code for. If you think this may be a biased opinion (it is fair to assume that), you can ask Hilaire Fernandes, and his experience in porting DrGeo from Squeak, then to Pharo, and then to Cuis, where it lives today.

BitBlit was still state of the art 25 years ago, and that was 20 years after you Dan invented it, maybe 45 years ago. In Cuis, instead of staying with a graphics model that is finally fair to say it is a bit outdated, or calling some external library (that would be perhaps the usual approach), we use a new, state of the art Vector Graphics engine, using newly developed techniques. It is written in Smalltalk and is now part of the OpenSmalltalk virtual machine.

The text layout and text editors from Squeak were enhanced and modernized.

A Smalltalk model for TrueType fonts, that is closely integrated with the VectorGraphics engine for performance. This is a big improvement over the StrikeFonts from Smalltalk-80, because they are scalable and high quality.

An implementation of Unicode, written 100% in Smalltalk.

Cuis is finally delivering its promises

It took some time, but:

Never forgetting the original values that defined Smalltalk-80,

Keeping in sight the long term objectives

While advancing step by step

Cuis was developed without commercial or financial pressure.

While it has been used by several companies, building commercial products, this never conditioned its evolution.

Still, having expectations from “serious” users has been positive, because it helped us focus also on reliability and performance.

If you were given a Smalltalk system that gives you:

- A GUI framework based on arbitrary, tangible objects
- That lets your app look great on any kind of display
- That eases the building of complex GUIs
- And lets you use code and user content in any natural language

What would you build with it?

So far, the only end user application taking full advantage of Cuis and Vector Graphics is DrGeo by Hilaire Fernandes.

But Cuis is only now becoming mature enough to build a next generation of interactive software in Smalltalk.

This is a great time to take a good look at it.

Contributors

These are some of the main contributors to the Cuis Project

- *Hernán Wilkinson*
- *Ken Dickey*
- *Mariano Montone*
- *Nicola Mingotti*
- *Andrés Valloud*
- *Jaromir Matas*
- *Nicolas Cellier*
- *Nicolás Papagna*
- *David Stes*
- *Philip Bernhart*
- *Dan Norton*
- *Casey Ransberger*
- *Hilaire Fernandes*
- *Luciano Notarfrancesco*
- *Gerald Klix*
- *Juan Vuletich*
- *David T. Lewis*
- *Eliot Miranda*
- *Gastón Caruso*
- *Nahuel Garbezza*
- *Phil Bellalouna*
- *Bernhard Pieber*
- *Masashi Umezawa*

20

Cuis has an active community of about 50 developers. Many of them contribute with bug reports, fixes, enhancements, questions and answers in the mail list.

These are the names of people who have done significant code contributions to Cuis.

We usually have technical discussions to reach consensus on technical decisions on our mail list. Also any issues that might affect external projects are raised, discussed and solved.

The community is friendly, and the traffic is reasonably low.
You are welcome to subscribe and say 'Hi!'

Some nice comments about Cuis

"Yay, Juan. You GO, guy! ...a great example of malleable software (and a clever mind) at work."

Dan Ingalls (Father of Smalltalk)

"I like it... It's nice and clean and simple and pretty. Nice stuff!"

Alan Kay (Father of the Dynabook, Leader of the Smalltalk and Squeak projects)

"I think you have a very elegant design aesthetic."

John Maloney (Creator of Morphic, for Self and Squeak)

"It's one of the smallest, definitely the fastest, and probably the best structured (Squeak) kernel that has been built. I very much enjoy Cuis."

Andreas Raab (Core contributor to the Squeak project)

"My favorite playful Smalltalk right now: Cuis. It is so small, yet has all of the coolest parts of Squeak and extends into areas like ML and vector graphics. It feels accessible and manageable."

Mark Guzdial (Creator of 'Media Computation' approach to C.S. Teaching)

"I am a big fan of Cuis"

John Sarkela (Long time smalltalker, Instructor, Professor)

I have been very impressed by the quality of Cuis. It is a master piece of craftsmen work, with a lot of love in small details.

Hilaire Fernandes (Developer of DrGeo, originally in Squeak, later in Pharo, now in Cuis)

"Cuis represents the best of Squeak: Elegant simplicity, high quality, and a sense of vision. It has a clean, crisp feel and is a pleasure to use. I really do appreciate Cuis."

David T. Lewis (Core contributor to the Squeak and Cuis projects)

"Aguante Cuis!"

Andrés Valloud (Long time smalltalker. Developer of Smalltalk VMs, Author)

"You are making a major contribution and Morphic 3 is a mouthwatering and urgently needed development."

Eliot Miranda (Leader of the OpenSmalltalk VM project, Creator of Cog and Spur)

"I like the reduced complexity of the image... It really feels like an unbloated Squeak image with the primary and important features still available."

Torsten Bergman (Long time smalltalker)

"If you would like to see Morphic done beautifully, check out Cuis. I simply cannot rave enough about how wonderful an experience it's been to work with."

Casey Ransberger (Long time smalltalker and contributor to Cuis)

21

Many of the people who inspire our work, are aware of Cuis and have praised it.

At the left you see their names, and the nice words they had for Cuis. Besides making me deeply proud, these are important because they mean, that to some point at least, we understood, what they tried to teach us.

It also means that Cuis is a worthy member of their lineage.

Cuis is also appreciated by the people who decide to use it for their projects, and to be active members of our developers community. On the right column, you see some of their names and what they think about Cuis.

That's all. Thank you.