Collaborative Praxis: The Making of the KeyWorx Platform

Sher Doruff, published in *aRt&D: Research and Development in the Arts*, Joke Brouwer, Arjen Mulder and Anne Nigten, Editors, V2_/NAI Publishers, Rotterdam, 2005

Introduction

There is no single methodology, no general description, that aptly depicts the making of a collaborative tool by a collaborating team. The process is as variegated as the personalities of the contributors and as fluid as the dynamic socio-cultural-economic ecology it inhabits. The working light that fills the lab on summer afternoons and streaks from desk lamps during the short days of winter, the proximity of workspaces to each other and the coffeemaker, shifting ambient sound landscapes, disappearing funding, aging equipment, new protocols, battles, brainstorms and broken cables all factor into the daily practice of working together. If one were to chronicle the hundreds of planning meetings, impassioned debates, demo disasters, external negotiations, the internal chaos and the nanoseconds of inspired synergy that make all the difference, the result would read pretty much as what it ultimately is – a sample cluster of purposeful social interaction in a creative context.

This project, spanning seven years, has naturally evolved along distinct strands. It began with the ambitious short-term goal of developing a multi-user collaborative tool for digital artists. Over time, it has migrated to an open-source technology platform, supporting the creation of potential multi-agent, multi-channel client applications across all sectors: the arts, education, business and service industries.

A Condensed History

In 1996, a group of four artists approached the fledgling Society for Old and New Media in Amsterdam to ask it to sponsor their proposal to create a tool that would enable interdisciplinary artists to collaborate in a virtual studio space on the Internet.[1] The proposal drew support from the Amsterdamse Fonds voor de Kunst and began to take shape, refine its goals and re-establish its team in 1997, under the guidance of Marleen Stikker and Carolien Nevejan, then codirectors of the SONM, which was renamed Waag Society for Old and New Media in 2002.

For the project, initially called KeyStroke, three programmers were recruited as programmer/developers: Tom Demeyer from Stichting STEIM[2] in Amsterdam, whose BigEye and Image/ine applications set an early standard for real time interaction; Just van den Broecke from AT&T and Lucent Technologies, who had expertise in multi-user technologies, and Niels Bogaards, an intern studying music technology at the HKU Hilversum. Each was to be responsible for the development of a component of the original framework – they were Realizer, Server and Patcher respectively. As the artist developer and project leader, I designed the user interface, defined and presented the concept to the public, test-drove the application in performance situations, facilitated workshops and drafted endless subsidy proposals. The programming team expanded over time to include Lodewijk Loos, Fokke de Jong (both music technology graduates of HKU Hilversum), Ben Soree (who studied graphic design at the Rietveld Academie Amsterdam), Eric Redlinger (a Brooklyn-based digital artist) and Arjen Keesmaat (of HKU EMMA). Klaas Hernamdt, Director of Operations of Waag Society, took over general project management in 2002.

As is often the case, the start-up funding was woefully insufficient for the task at hand. In the late 1990s, support for nonprofits engaged in technology development for the arts was little understood, and validation of the "art of code" was still hotly contested.[3] With brave conviction, the Waag Society, a nonprofit cultural institution, remained committed to the project, digging deep into its own pockets to sustain its development. That commitment has paid off in many respects, as the technology is now the transparent framework for a large percentage of Waag Society projects.

For the first three years of the project, the small core team of four people worked part time, one or two days a week, with the exception of Bogaards, whose full-time internship and eventual employment provided a vital cohesiveness. But there was an inherent splintering in the development process, in the modular design itself and the fabrication of it. The contingencies of the working-alone-together model that was pragmatically adopted elicit interesting analysis within the larger context of collaborative process.

So, too, the emerging network of contributors forms a community structure. Artists interested in the concept of synchronous collaborative spaces on the Net have been critically important as members of the extended team. Their continued alpha and beta testing, often under stressful performance conditions, and their suggestions for new plug-ins and design improvements have advanced the R&D beyond the vision of the Waag Society developers. The ad-hoc contribution of the artists/end users has been, and continues to be, essential to the development of the tool[4], in many ways redefining the very notion of R&D teamwork.

What Is It?

The KeyStroke[5] application was designed for multi-agent, real time, distributed, synchronous performance by new media artists. In effect, it is a tool for multimedia, telepresence experimentation and jamming. It was built for the Macintosh OS utilizing C++, Java and XML languages. In 2003–04, the Waag Society began merging KeyStroke with its KidsEye framework to build an extensible platform for a wide range of multi-agent functionality, synchronous and asynchronous. Programmer Ronald M. Lenz (of the chemical engineering department, TU Delft) contributed the integration of a content management system implemented in the KidsEye[6] project, and Van den Broecke designed a user authentication system for managing agent permissions across individual spaces. ScratchWorx[7], a spin-off hardware interface designed by HKU EMMA students with technical support from Waag Society, further extended the reach of the platform to a youth target group. Early 2004 saw the completion of the open-source platform under the Mozilla Public License, GNU General Public License and GNU Lesser General Public License triple license agreement.

The original KeyStroke/KeyWorx combined a distributed multi-agent, multi-channel environment with dynamic cross-media synthesis, providing the tools for extensible forms of telecommunication, telematics/telekinetics, interactive broadband and collaborative performance. Its ability to synchronously synthesize media in a translocal workspace makes it a powerful live performance tool for remote interaction and interdisciplinary work. The new open-source KWart (which will replace the KeyStroke/KeyWorx protocol) will add the possibility of asynchronous content management (CMS) to the functionality. A description of the platform is beyond the scope of this article. For more information see the white paper by Van den Broecke[8].

Collaborative Process and Design Approach

The working-alone-together praxis – a modular method for a modular architecture – that this team adopted in the early years of the project for pragmatic, financial and personal (i.e., long-established individual working patterns) reasons represents a curious corollary to certain aspirations of distributed collaborative learning and performance software environments. At issue is the method of recursive collaborative process; specifically, how the KeyWorx team "collaborated" on a technology designed to enable "collaboration" between interdisciplinary artists and how those production methods and values are interfaced to the end users.

That the team feels the development process would have been enhanced by working in close physical proximity is evidence of the learning curve immanent in quality remote interaction. It is also evidence of the infrequency of the collective use of the application by the makers. This implies that not all developers have a stake in using the technology they're making, and that programmers and developers evaluate their work from different microscopic lenses. When polled, in 2002, the KeyWorx team unanimously felt it would have expedited the project if they had all worked in the same room. That's a poignant remark about a project focused on facilitating remote collaboration. Ostensibly, though, the programming process is dissimilar to the user experience of the software, so a strict corollary is misguided. Interestingly, the process of enlarging (through the addition of CMS) and open-sourcing the platform from 2002 to the present has meant a change in the lab configuration, working conditions and methods. Programmers mainly work full time and in earshot of one another, with a fairly conventional project management overview. Lenz and Van den Broecke have adopted an extreme programming approach to the expansion of the platform. This metamorphosis is reflected in the open-source product. Lenz explains that in talking through what you're doing together, "you don't necessarily get better code, but you do get a better architecture."

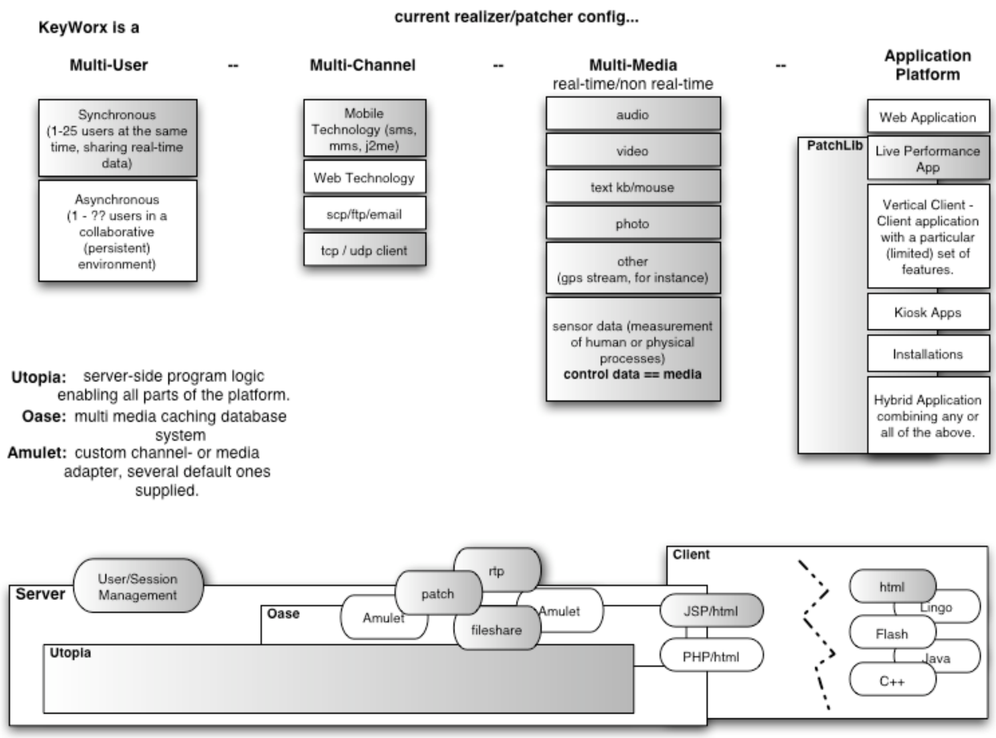
KeyWorx is a
current realizer/patcher config...
Multi-User
--
Multi-Channel
--
Multi-Media
real-time/non real-time
--
Application Platform
Synchronous (1-25 users at the same time, sharing real-time data)
Asynchronous (1 - ?? users in a collaborative (persistent) environment)
Mobile Technology (sms, mms, j2me)
Web Technology
scp/ftp/email
tcp / udp client
audio
video
text kb/mouse
photo
other (gps stream, for instance)
sensor data (measurement of human or physical processes)
control data == media
PatchLib
Web Application
Live Performance App
Vertical Client - Client application with a particular (limited) set of features.
Kiosk Apps
Installations
Hybrid Application combining any or all of the above.
Utopia: server-side program logic enabling all parts of the platform.
Oase: multi media caching database system
Amulet: custom channel- or media adapter, several default ones supplied.
Server
User/Session Management
rtp
patch
Oase
Amulet
fileshare
Amulet
Utopia
Client
JSP/html
PHP/html
html
Lingo
Flash
Java
C++

Diagram of the KeyWorx architecture

The Programmers' POV

The perspective of the programmers is often overshadowed in discussions of artist software and technologies by the aesthetic aims of the development. Most developers would agree that the style and design of the back-end code is every bit as "aesthetically relevant" as the front end. I interviewed the three core programmers of KeyStroke/KeyWorx individually in 2002[9] and asked them questions about the development process in relation to the desired functionality of the application. I posed questions concerning the mirroring of the development methodology in the product, and how and where the collaborative development process – the making together – is evident in the design, functionality and aesthetic of the application. I asked if the strengths, weaknesses, idiosyncrasies, styles and philosophies of the development team were present in the functionality of the tool and the way it interfaced with the user. Here, I include a selection of answers regarding the collaborative development process as well as a reflective comment by Marleen Stikker and myself.

Tom Demeyer

SD: How is the development process mirroring what we now have in KeyWorx?
TD: The process is mirroring the characters of the people, mirroring the code. We're solitary programmers (well, Just van den Broecke less so). We've always programmed by ourselves; we've never been part of a big team of programmers – this is just a clever trick for avoiding that. We just do our own thing, while still being part of a team. If we really worked together the whole time, we wouldn't have as many problems as we do in the multi-user department, because we'd be testing much more while we're doing it. [But] since we're working alone, you may have this good idea and work a long time, but when it all has to come together it fails, and it turns out that it wasn't ever possible. I think this might be the central disadvantage to working this way.
SD: So if you're working on a multi-user application you should be developing in a multi-user environment yourselves?
TD: Yes. Or you have to set a very rigid multi-computer environment for yourselves. When you write code, you're a very casual user as well, on submicroscopic bits of the code. You're using it to test the validity of the code, the assumptions, but you're using it microscopically, not anything like an end user would use it. It's just small tests. In a multi-user bit of code it's not possible to do it yourself. In these kind of programming environments like the Waag, it's always true that the process also reflects the character of the people, the way they work together – it all reflects in the end product, there's no question about that. That's why you won't ever see big corporate structures writing innovative software.

Just van den Broecke
SD: How is it that you three programmers have collaborated on a piece of collaborative software? How much is that process reflected in KeyWorx? In this particular case you have a collaborating group of people working on an application that's about collaboration.
JvdB: For one thing, the multi-user aspect has been greatly underestimated, and for some people it's something like a side artifact. For example, to the Realizer, to put it bluntly, it's another interrupt. I had the same thing when I started working on multi-user, because I came into this group at AT&T and moved internally to another group (working on multi-user), and said, "What's the big deal? You send some stuff to each other. Why are you working with hundreds of people on this with these thick documents?" But when I started working on it, I realized, for one thing, that what I underestimated was the number of failure modes that can arise. At that time we developed user interfaces as a side effect to access the system, [rather than]

starting with users first. Maybe that's what also happened here. There were already so many issues to solve. And for me, from a network point of view, I tend to develop from the inside out. But to answer your question, I think the approach is reflected in the product.
SD: With this particular application, I wonder if there shouldn't have been more play, or interplay, from the very beginning amongst the team?
JvdB: That's also something that happens a lot with programmers – they make something for someone else. But again, this is also a criterion for good architecture: eat your own dog food. Would you use this yourself as a product? I think our intention is to do so.

Niels Bogaards
SD: Would it have been better to be working in the same space?
NB: Sure. If you're all in the same room, or at least at the same level of intensity, then you can also excite each other. Whereas now, everyone has their island, and you don't know what's happening on the other island, more or less, and then you're unsure if things that happen in the other parts are sound, even. It leads to a situation where you think, "Maybe it's better if I do that."
SD: How does the development process effect a KeyWorx session? Does your frustration reflect in the design of the application?
NB: I don't have the same feeling, because it's not the same people I'm in the KeyWorx sessions with, and very often they're at the same or higher level of intensity than I am, so that doesn't really apply. I still very much agree with the ideas behind the application. I think almost everything we decided after one year of thinking is still true ... I'm not at all disappointed with the genre or working with the application. It's the most exciting thing I know of. I know that I can have real fun in KeyWorx, and I know that others can.

Comment by Marleen Stikker
This is the only project where people are really angry about certain decisions about software. I really love it. That makes so much sense. To me it's what it we are all about: that choices in software are not objective choices, that they're subjective and exclude certain possibilities and enable other possibilities. That's my interest and why I'm doing this. To me, the KeyWorx team is the living example that software is culture.

Conclusion

My semantic relationship with the term "real time," has profoundly transformed since 1996. In the mid-1990s, "real time" interaction promised the extraordinary opportunity to dynamically process media states without rendering times that approximated the lifespan of an insect species. Now "real time" is a ubiquitous, transparent functionality, and the term itself is nearly obsolete. My current relationship with this term has more to do with questioning what we thought we meant by the concept of "real" "time" in the first place. It is certainly a technologically driven term with philosophical challenges. My artistic fascination with the implementation of synaesthetic algorithms has steadily migrated to a fascination with the intersubjective experience of human-machine-human interaction in distributed, creative environments. I believe there is a reluctant acceptance of theoretical research within the R&D lab positioned towards rapid prototyping. Art and science collaboration is a much discussed topic, but I would love to see philosophy thrown into the mix in a very practical, everyday sense. KeyWorx, like any other software generated in an arts context, has an aesthetic character. Much of the work that I've witnessed from artist performances has a montage quality in which the individual contributions become indiscernible in the cooperative product. I have also seen people design multi-user games, theatrically situate public-space surveillance cameras, broadcast local soundscapes from walkie-talkies, and project collective protest messages onto buildings during the recent Republican National Convention in New York. Aesthetic is as aesthetic does. In my view, the role of KeyWorx as a tool is best expressed by Felix Guattari when he argues that we are autopoietic machines that self-produce worlds in the ethico-aesthetic paradigm and must take responsibility for the creative instance that produces the created things.

That collaborative efforts cannot be generalized about and are subject to the chemistry of personality, the whims of mood, the weather and the price of milk is pretty much agreed. Sustained collaboration builds upon a fragile foundation and formulates degrees of trust, weighted probabilities, and shared outcomes. The intersubjective dynamic, deep convictions and anarchistic attitude of the original KeyWorx team are embedded in the behaviors, style, function/dysfunction, language and philosophy of the technology.

For example, there was an early conscious decision NOT to implement a moderator/director function in the application, requiring players in each

KeyStroke/KeyWorx session to negotiate or abandon hierarchical structures. There is more than vision, code and design here. There is a mini-culture, at once elastic, cryptic, resilient, stubborn, impatient, impulsive, cautious and enthusiastic. If there is a truly bottom-up, processual aesthetic to be identified with the technology – the distributed working-alone-together of the participating artists – it is built upon the dynamic articulation of cooperative strategies from which the creative may emerge.

Additional KeyWorx functions 2003-2005

Two important functions were augmented in the application in 2003 - the Google imagecrawler module and the SMS module. Both plug-ins, within the structure and organization of the application as a system can be viewed as hyperautonomous affects - "They are autonomous not through closure but through a singular openness. As unbounded "regions" in an equally unbounded field, they are in contact with the whole universe of affective potential, as by action at a distance" (Massumi, 2002a, 43). The imagecrawler, a simple bot filter in KeyWorx, opens the enormous Google database of images to players during a jam. By instantiating the crawler and typing in a word or url, the images are displayed in the performance at the speed, size, screen position, etc., discretion of the players. Though not a random function, as Google's database is hierarchical (based on popularity), there is a random feel about it, an unpredictability. In itself it mimics Bergson's center of indetermination, selecting usable images from the universal flux of images. As these images, upon entry into the playing field, can be modified by other processing operations it also mimics Hansen's digital upgrade of Bergson's image selection by filtering and creating new images.

The inclusion of an SMS text messaging module enabled local and nonlocal participants (audience) to contribute text images to the performance, further opening (structural coupling) the autopoietic recursivity of the KeyWorx system to communications systems outside the Internet. The structural architecture of KeyWorx is extensible. Its organization is affective, simultaneously bringing the outside in and the inside out. As a structure, KeyWorx could be described as topological with an ontogenetic and autopoietic organization. An abstract machine.(see Chapter Six). Both of these functions played an important role in the Interfacing Realities Connected Event examined in the next Hinge.

And finally, as we begin to look at vitualities, durations, perceptions, memories, intuition, it's interesting to wrest one misnomer, relative to this thesis, from the technological semantics of the application. The engine in KeyWorx that renders all the media formats in real time is called the "Realizer." It is a separate component in what was a three-part architecture - Server, Patcher, Realizer - from 1998 until 2004. (The Patcher is reconceived in the open source application as the TCC – Traffic Control Center. The original Realizer had a separate output window that displayed the composited performance. The name of this engine was christened by programmer/developer Tom Demeyer. Were we constructing a fiction or a supple, evocative hindsight, we would better call this engine - The Actualiser.

The common misconstrual in new media discourse and software engineering, is considering the virtual to be a simulation of reality. Hence, Demeyer's Realizer makes "real" all "possible" algorithmic re-presentations. As we have stressed and will continue stressing, the power of the virtual is its realness. Within the affective interplay of a KeyWorx event, the always already real/virtual is actualised, in one of many dimensions, in the composited real time artefact of the audio-visual output. Distinctions between real, possible, actual and potential will be discussed in the following chapter.

[1] David Garcia, Barbara Pyle, Nanette Hoogslag and Sher Doruff.

[2] http://www.steim.nl.

[3] KeyStroke received support from Stichting STEIM, het Amsterdams Fonds voor de Kunst, de Mondriaan Stichting, Fonds voor de Podium Kunst, The Arts Alliance (UK) and MultiMediaLab2 (UK).

[4] Artists such as Michelle Teran, Jeff Mann, Motherboard, Ellen Roed and Isabelle Jenniches.

[5] For clarity, KeyStroke, the original application targeted to artists, was renamed KeyWorx in 2002 when a company designing keystroke recorders challenged our right to the name. KeyStroke, the artist application, became KeyWorx for a time. The protocol for the old KeyStroke/KeyWorx will be phased out by the end of 2005, leaving the new open-source reference application to be renamed once again to KWart – inelegant but functional. KeyWorx is now the name of the extensible open-source platform from which a wide variety of clients can be built – by anyone.

[6] http://www.waag.org.

[7] Ibid.

[8] http://www.keyworx.waag.org.

[9] The full interviews are available at
http://www.smartlabcentre.com/radical/good_practice/doruff.html.