LITERALLY UN-LOCK AND SPEED-UP YOUR CONTAINERISED DEVELOPMENT ON EMBEDDED DEVICES
by example of a standalone gadget with orientation sensors sending Open Sound Control

Martin Carlé

PhD Candidate, Department of Audio and Visual Arts, Ionian University,
mc@aiguphonie.com

Abstract
The application in and to the Internet of Things mentioned in the subtitle demonstrates a case study of how Literate Programming strategies based on Emacs' org-mode can be generally employed to break free from limiting, often vendor imposed patterns in modern, all so 'eazy' and ever more container-oriented software development. It shall be argued that by adopting the proposed method, the typical cycle of building, deploying and updating software on embedded devices will not only gain in speed, transparency and freedom but will also reach out to new frontiers of rapid prototyping and add extra value to the domain of education and creative experimentation.

Keywords: media literacy, IoT, Linux containers, literate programming, Docker, creative technologies, e-Learning.

1 Introduction
Talking and reporting at a conference on 'interdisciplinary creativity' about the beneficial con- sequences of a tiny and rather specific application in the vast domain broadly identified as the 'Internet of Things' (IoT) asks for a contextualisation of motivation and a clarification of terms. Now, if this encompassing domain—or even our 'world'— of the IoT can be defined as the manner of (i) being connected (ii) by smart devices (iii) everywhere (iv) all the time, then any application to the organisational realm of this fourfold of being must be expected to induce fun- damental changes to our entire media infrastructure. If, moreover, the present infrastructure of information processing is defined by the corresponding tetrad of technology, namely (i) by a het- erogeneous grid (ii) of cyber-physical devices (iii) and distributed services (iv) cooperating in near real-time, then what we used to call 'software' nowadays constitutes this very organisational realm that is governing our overall mediation and, consequentially, is shaping our existence in the world. Programmable media, to put the given ontology differently, even if conceived as being controlled by a human written 'software' or if regarded as 'physically' manifested by embedded devices right at hand, are nonetheless no more things at our disposal like pencils or screwdrivers were.

In contrast, they do act by themselves and, therefore, can neither be under- stood as tools like the traditional humanist's stance still holds, nor can they be conceptualised as 'facilities for creativity' extending our brains or bodies in the vein of McLuhan (1964). Media not merely reveal and decide upon what presence means to man, they have grown—through their sheer ubiquity and indispensability—to an entity of their own.

Only such an updated ontological perspective (Heidegger, 1950/1977), and thus more com- prehensive view, allows us to recognise and further cognise what the quite recent 'containerisa- tion' of software and services actually implies and why this event amounts to nothing less than the most significant media upheaval that we are currently facing. Since, however, There is no Software (Kittler, 1995) anyway, but […] there are just Services (Kaldrack & Leeker, 2015), the respective media-theoretical viewpoint which follows from these denoted standpoints may not only remind us of the principle fact that any sensing, transmission and processing of information depends on restrictions and possible interceptions imposed by the underlying hard- ware, but shall also make us understand that the systematic—and not at least historical—piling up of software layers and language levels along with the inevitably resulting logical 'sponginess' and insecure 'porosity' is running that ubiquitous discourse network which intervenes ever more in our social discourse, political opinion formation and disposition towards knowledge.

Considering, furthermore and more locally concrete, that until a last year's presentation, 'containerisation' proved to be an alien concept at the Department of Audio and Visual Arts and that Docker Inc.—this striking "cloud 'container' company" (Fortune.com, 2015) growing even faster than Amazon.com Inc. or than Google Inc. ever did— was an unheard name, it becomes likewise inevitable to point out, at least briefly, what merits and drawbacks a container- ised application development methodology brings about in general terms, and in particular, how it concerns the field of education and an "Interdisciplinary Creativity in Arts and Technology" (DCAC, 2018). Because given the above outlined media condition, 'creativity' in both, an artis- tic as well as in a technological sense, cannot mean anymore to simply 'use' some all-so-smart and ready-made devices in order to deliver some arty 'content' or to supply some fancy 'con- ceptual media artwork' to be soaked up by the banalising cultural networks and their secondary market interests in innovation or mere communication. Creativity today rather must entail to insert actual processors and algorithms into the very realm that reshapes presence as such and thereby our standing in the domain of the internet of things—if creativity, at last, shall lead to more than "σύγχρονες εκφράσεις"[1] (Γιώργος Γραμματικάκης, 2017).

As a consequence, everything starts from gaining an appropriate computer literacy. As such, the introductory section shall be closed by referring to the remarkably creative and educative project Origami Singing of Manty Albani (2018) exhibited in parallel to this conference at the 12th Audiovisual Arts festival for her dedication of bringing embedded devices to children living in remote villages at the island of Corfu in order to practically familiarise them with the elementary and so unerringly taken up media-triad of sensing, transmitting and processing as stated by the small bill next to the final work which made her intervention that perfectly clear:

[…] Sensors of a "BBC micro:bit" micro computer, attached to the Origami collect information which is then transmitted and processed into sound.[2] […] the project is part

of a theoretical framework that includes S.T.E.A.M (Science, Technology, Engineering, Arts, Mathematics), the "D.I.Y." and "maker culture" movement. This attitude also involves a political opposition to the dominant cul- ture of mass consumption and uniformity.

2 Coding cultures, virtualisation and Linux Container Technology

2.1 Grabbing code and pasting it on embedded devices has never been that 'eazy'

What has to be appreciated as pioneering steps towards a sound foundation in computer lit- eracy at elementary schools when carrying out first steps of programming cyber-physical de- vices via a "JavaScript Blocks Editor" or a virtualised dummy device through an always ready MicroPython web portal invitingly calling for "Let's Code" (Micro:bit.org, 2018a), such an "Easy Peasy" approach of the British Broadcasting Corporation proudly advertising "no software required",[3] or a continued reliance on Arduino's celebrated "easy-to-use hardware and software" (Arduino.cc, 2018a) not even able to run a proper language interpreter on their "playground" of micro controllers (Arduino.cc, 2018b), cannot remain the model for graduate students of a university's media arts department. Equally, if uncritically buying some MicroPy- thon-enabled chips off-the-shelf now available, we are staying behind. Since always having been save-guarded by closed environments in the consumer's comfort zone, we will have never learned to creatively step out of our pre-manufactured padded cells and to cope with neither macro nor micro 'computers' of the real world deserving the name and ruling the game.

Saying this, however, is not to insist, like any true Linux nerd most probably would, that proper computer literacy only begins, if you have managed to compile your first custom kernel or device driver. On the contrary, as shown by example of the New Out Of the Box Soft- ware installer NOOBS (raspberrypi.org, 2018a), far more capable circuit boards in about the same price rage that are really in line for becoming "Your Next Linux PC" (Hartley, 2015) al- low any tinkerer to freely choose between a pretty wide range of almost entirely Linux-based operating systems (raspberrypi.org, 2018b), each specialising on another flavour of interest and individual needs to reach out for "insanely innovative, incredibly cool creations" (PCWorld,

2018). Linux, thus, proves once more and now in the domain of IoT to be the true source of flexibility and freedom of the 'in-dividual' in all literal sense. Sustainably boosted by the cod- ing culture of Free/Libre and Open Source Software (FLOSS) (Stallman, 2013/2016), it firmly remains as the only reliable gateway to mature creativity. Hence, the actual disgrace of fostering incompetence and inclusion through an addictive 'eaze of use' and coddling comfort is rather to be detected where the versatility and openness of Linux tends to be disguised or is about to be perverted, if not abused.

That threat is given, if the overwhelming and thereby double-edged 'delivery of all wishes' is served by the 'gift' to simply skip any involvement in community- based coding, but instead to grab everything ready for use from free image repositories encompassing the whole range from a huge variety of pre-made system distributions, over diversely pre-configured software packages, until fully pre-checked and practically approved services. This heavenly and false paradise at once, though, has already become

a reality for the cloud business by means of what we are focussing on: containerisation. Accordingly, the market leader's promise, finally becom- ing irresistible for anyone, from the individual SysOp over the DevOps departments until the software management headquarters, plainly bids:

"Build, Ship, and Run Any App, Anywhere" (docker.com, 2016).

Yet, why shall this heaven for the high cloud biz concern the earth-bound D.I.Y. maker and sober students of media art, theory and engineering? Well, what other is Docker's marketing slogan than the ubiquitous fulfilment of the aspiration implied by the term: The Internet of Things? What then, since we already run Linux, should prevent that this reality for today's services in the cloud shall not become true on the embedded earth of tomorrow, meaning for the building, deploying and updating of software on and for tiny cyber-physical devices just as well?

Still, the decisive difference between shipping 'dockered' containers and dealing with codes and systems the 'Linux way', is finally to be articulated by words like 'transparency' and 'free- dom'. But, the advantages in efficiency and the enrichment of possibilities are undeniable and too powerful to fight against or to do away with them by moral reservations. The general quest, however, what's going on within our media—as now with containerisation—is not and never foremost an ethical one, like when asking: How only could this have happened to us? The media-theoretical question and starting point for what can be done about something, is rather and always: What made all this so inevitably necessary?

In preliminary terms, the answer has already be given. At the bottom line it is the pressing need to slim down what above has been called the logical sponginess of software and language layering from which the promise to raise efficiency and to seal-off the sponge's porosity in soft- ware development and service management took its drive to accumulate a never seen stream of venture capital investment making Docker—while building on Linux Container Technol- ogy (LCT)—the shifter of a paradigm. The paradoxical irony of this shift lies in the successful strategy not to decrease layering, as one may expect, but to increase it by an enforced granu- larisation of virtualisation. The perversity about the achieved solution, not merely being ironic, is that—ostensibly for security reasons—any pile of docker containers gets dependent on an all-pervasive daemon process called dockerd which seriously undermines the flexibility and independence of your containerised applications on Linux. Accordingly, the media-theoretical answer is: privatising "Integrated Security" sells (docker.com, 2018). Again, we have reached the stage where everything starts from computer literacy, since there is no way not to know upon which technological grounds each time a certain power structure has been build.

2.2 Foundation and prospects employing containers on embedded devices Now, the strategy of virtualisation is nothing new at all; neither is the encapsulation of processes to achieve it. If an isolated filesystem gets added, such an aggregation is called either a system container or an application container. Each instance of the former, as possible with Linux Containers (LXC), runs a fully virtualised Linux operating

system, whereas the latter, as spawned by dockerd, is just an ordinary process with

separated namespaces (pid, net, mnt etc.) and resource limitations (CPU, memory, block I/O. etc.) controlled by cgroups. Control groups reached the mainline Linux kernel 2.6.24 by 2008, namespaces date even back to 2.4.19 in 2002. But full container virtualisation relies on user namespaces, only finished by release 3.8 in February 2013. In fact, LCT was ready one month before docker has been intro- duced by Solomon Hykes at his French company called dotCloud. Setting out as an high-level feature-wrapper around LXC, insulation was soon achieved by docker's own libcontainer.

In contrast to full-fledged virtual machines depending on total hardware abstraction by an hypervisor, containers running directly by the kernel are obviously far more lightweight mak- ing them eligible for embedded devices in the first place. In effect, experimentation with slim custom kernels and ports of dockerd for several ARM-based architectures begun almost im- mediately. Among the most matured resinOS (balena. io, 2013) and hypriotOS (hypriot.com, 2015) are to be named. Yet, also the pioneering, all docker-based environment gadgetOS with the accompanied command line interface (CLI) gadget developed by Next Thing Corpora- tion (NTC) is worth mentioning for targeting their extraordinary feature-laden, very low-costs and super tiny C.H.I.P. Pro product, bringing thus, for the first time, truly wearable wireless audio projects in sight (NTC, 2016). At latest since Docker Inc. officially supports Paspberry Pi (docker.com, 2017), all sorts of applications may stimulate the makers´ imagination, reaching from experimentation with a fleet of distributed networked sensors and effectors until inter- and intranet service prototyping scenarios on affordable local cluster arrangements.

All these applications have one thing in common. They are closing an imaginary gab between the big cloud biz and low-budget tinker projects, namely by the common need for orchestration. That is essentially the frequent, repetitive and thus boring building, deploying and updating of software for a possibly heterogeneous grid of computing devices, and secondly an optionally au- tomated scheduling, scaling, load-balancing and monitoring of container instances. Here again, at the core—notwithstanding the unmatched interface to free multi-architecture image reposito- ries or industrial orchestrators like Google's Kubernetes offered by Docker—there, where we depend on dockerd to build our own application container first and foremost, we face the essential bottleneck of the overall system. Especially when the size of images really matters, like with embedded devices, building and updating containers the 'Docker way' gets pretty cumbersome and painfully slow. Fortunately, there are rivals trying, on the one hand, to liberate the launching and controlling of containers the 'Linux way' by an init system, like systemd, on the other hand, to undertake the standardisation of container formats such that building them can be more efficient and secure from the outset, like both is currently pursued by rkt of CoreOS. Yet, for the time being, the lessons learned with gadgetOS during the project mentioned in the subtitle, the strategy applied and solutions found, shall be briefly described next.

3 Literate Programming and software orchestration

The point of intervention, thus, is to break free from the monolithic dockerfile format speci- fying the container building process. The gadget CLI accesses it, while relying on the docker API, through an almost incredibly easy to use triad of commands in order to get your application on the C.H.I.P.: build, deploy and run. The strategy to

literally un-lock, modularise, secure and speed-up your containerised development is to employ the capacity of Literate Program- ming (LP) to make any coding more flexible and transparent. Although LP appears to be an age-old concept, originated by Knuth (1984), the father of TEX, its most powerful implementa- tion by org-babel for Emacs (orgmode.org, 2018) has extended its applicability considerately. By means of the classical LP methods called 'tangling' and 'nowebbing' org-babel allows to extract and connect arbitrary and parametrisable code-blocks of diverse languages on the fly, such that they 'literally' transform into a dynamically created chain of executables. Hence, as a methodological motto, we may say, if literate programming applied to code- writing means:

"Don't comment—write a book!", then, in the days of software container shipment, this maxim translates to: 'Don't trust the container ferryman—tangle your own orchestration!'. The solu- tions achieved following this proverb, can be outlined as follows:

1. dynamically 'tangle' the central yaml config file, rather than building the dockerfile
monolithically by the gadget CLI
2. 'no-web' the necessary UUIDs directly from a parametrised docker go-lang executable
3. safely incorporate private git-repositories during container build-time rather than expos- ing your private ssh-key to a potentially public dockerfile.
4. modularise and rearrange the container building process in such a way that the resulting image can still be shrinked to fit on the C.H.I.P.

The corresponding code and explanations which would never have fit on six pages can be re- trieved at Carlé (2018) along with a full set of instructions to reproduce the results. Or, if you prefer it the eazy way: get docker for 'free', grab my pre-made image and simply execute deploy & run.

4 Conclusion

After awareness was raised during the introduction for the current media condition as living by a double set of four, the 'fourfold of being' and the 'tetrad of technology', that in short may be called the Tetraktys in Square of our times in a world of the Internet of Things, focus could be set on the hot-spot of containerisation resulting in an irresolvable double-sidedness of the ease of use of software tightly intertwined with the value of education and a meaningful refer- ence to creativity. The general lesson learned from a particular pilot project employing the sole production-ready container technology at percent, revealed that a fair balance of the involved merits and drawbacks can be achieved in general, by enforcing transparency and freedom of use, in particular,

by an adaptation of the respective time-proven concepts of Literate Program- ming. As a result, container building and deploying becomes significantly faster, truly easy, literally teachable, more secure and flexible, such that new frontiers of rapid prototyping and creative experimentation are pushed open. The example application originated with a commit- ment to the Performance Environments Research Lab (PEARL) of the Ionian University contributing to a sensor-laden dance project to be synchronously performed around the globe (Zannos & Carlé, 2018). In such cases, given the distributedness of devices, the need for an agile management of small teams with varying tech-savviness and a hot loop of software development during rehearsals, containerisation even amounts to an 'enabling technology'.

References

Albani, M. (2018). Origami Singing / Ηχητικό Οριγκάμι. Retrieved October 30, 2018, from https://users.ionio.gr /~a16alba/M_Albani/portfolio-1-col_multimedia. html

Arduino.cc. (2018a). Arduino - Home. Retrieved October 31, 2018, from https://www. arduino.cc/

Arduino.cc. (2018b). Arduino Playground - Python. Retrieved October 31, 2018, from https://playground.arduino .cc/interfacing/python

balena.io. (2013). balenaOS - Docs. Retrieved November 3, 2018, from https://www. balena.io/os/docs

Carlé, M. (2018). Gadgets / bno055_cpio_osc_py2 · GitLab. Retrieved November 4, 2018, from https://gitlab.com /chip_gadgets/bno055_cpio_osc_py2

DCAC. (2018). Digital Culture & AudioVisual Challenges. Interdisciplinary Creativity in Arts and Technology - CALL FOR PAPERS. Retrieved May 24, 2018, from https://avarts.ionio.gr/dcac/2018/cfp/docker.com. (2016). Docker Cloud - Build, Ship and Run any App, Anywhere. Retrieved November 2, 2018, from https://cloud.docker.com/ docker.com. (2017). Get Docker CE for Debian. Retrieved November 3, 018, from https://docs.docker.com/insta ll/linux/ docker-ce/debian/docker.com. (2018). Security | Docker. Retrieved November 3, 2018, from https://www.docker.com/products/sec urity

Fortune.com. (2015, April 14). Docker, a cloud 'container' company, raises $95 million. Retrieved October 30, 2018, from http://fortune.com/2015/04/14/docker-raises-95-million/

Hartley, M. (2015, June 1). Raspberry Pi As Your Next Linux PC. Retrieved October 31, 2018, from https://www.datamation.com/open-source/raspberry-pi-as-your-next-linux-pc.html

Heidegger, M. (1977). The question concerning technology, and other essays. New York. (Original work pub- lished 1950)

hypriot.com. (2015). About Us · Docker Pirates ARMed with explosive stuff. Retrieved November 3, 2018, from https://blog.hypriot.com/about/

Kaldrack, I., & Leeker, M. (Eds.). (2015, September 21). There is no Software, there are just Services. Lüneburg. Kittler, F. A. (1995, October 18). There is no software. ctheory, 10–18. Retrieved from http://www.ctheory.net/articles.aspx?id=74

Knuth, D. E. (1984, January 1). Literate Programming. The Computer Journal, 27, 97–111. doi:10.1093/comjnl/27.2.97

McLuhan, M. (1964). Understanding media: The extensions of man. London.

Micro:bit.org. (2018a). Let's Code | micro:bit. Retrieved October 31, 2018, from https://microbit.org/code/ Micro:bit.org. (2018b). Meet micro:bit | micro:bit. Retrieved October 31, 2018, from https://microbit.org/guide/ NTC. (2016). Get C.H.I.P. and C.H.I.P. Pro - The Smarter Way to Build Smart Things. Retrieved November 3, 2018, from https://web.archive.org/web/20180714130046/ https://getchip.com/pages/chippro orgmode.org. (2018). Babel: Introduction. Retrieved June 30, 2018, from https://orgmode.org/worg/org-contrib/b abel/ intro.html

PCWorld. (2018, March 16). Raspberry Pi projects: Insanely innovative, incredibly cool creations. Retrieved Oc- tober 31, 2018, from https://www.pcworld.com/ article/2895874/computers/10-insanely-innovative-incred ibly-cool-raspberry-pi-projects.html

raspberrypi.org. (2018a). Download NOOBS for Raspberry Pi. Retrieved October 31, 2018, from https://www.ras pberrypi.org/downloads/noobs/

raspberrypi.org. (2018b). Raspberry Pi Downloads - Software for the Raspberry Pi. Retrieved October 31, 2018, from https://www.raspberrypi.org/downloads/

Stallman, R. M. (2016). FLOSS and FOSS - GNU Project - Free Software Foundation. Retrieved October 31, 2018, from https://www.gnu.org/philosophy/floss-and-foss.html

Zannos, I., & Carlé, M. (2018, July 4). Metric Interweaving In Networked Dance And Music Performance. doi:10 .5281/zenodo.1422665

Notes

1. Trans. "contemporary expressions". Giorgos Grammatikakis was Chairman of the Ionian University and re- sponsible for founding the Department of Audio and Visual Arts.

2. This part of the work was realised with SuperCollider running on a an Raspberry Pi and has been carried out by Vasilis Agiomyrgianakis.

3. Micro:bit.org, 2018b: "Easy Peasy: It can be coded from any web browser in Blocks, Javascript, Python, Scratch and more; no software required."