

INNOVATING@SUN

PROJECT LIVE *

Announcer: You're listening to the Sun Microsystems Podcast Network. Welcome to another edition of Innovating@Sun with your host, Hal Stern. Today's topic: Project Live. And now, here's Hal Stern.*

Hal Stern:

Hello and welcome to another episode of Innovating@Sun. I'm your host, Hal Stern, Senior Vice President of Global Systems Engineering, and I'm joined by Olaf Manczak, who is a Senior Staff Engineer in Sun Labs, and the principal investigator of something called Project Live*. So, Olaf, welcome to the show.

Olaf Manczak:
Hello.

Hal:

And I guess, tell us a little bit about what problem you're trying to solve with the Live* Project.

Olaf:

Practically all of today's software that runs on consumer devices is based on operating systems that we use also on data center servers, with wireless boxes that run Linux. We have Sony LCD TVs that run Linux, and when you deal with these consumer electronic devices, you never have to deal with updates, patches, etc. Updates to a Mac OS on a MacBook require every user to accept and install some updates, reboot afterwards. Updates of the same Mac OS on an iPhone are much simpler; we simply download new versions of firmware, restart the device with the new version, and it basically works out of the box.

Hal:

Olaf, I think the emphasis on thinking of things as firmware, thinking of a firmware level, is really important, and made eminently clear to me just last week when I put the latest security patch onto my iMac and found out that it broke the CUPS printing system that I had installed with an older version of the HP All-in-One printing software. So, clearly, a bit of a fuzzy boundary there led to some unpredictable behavior that was visible to the user.

Olaf:

Yes. So, I believe in consumer electronics, the main advantage of using firmware model is that it gives, not only simplicity of use for the end user, but it also makes the behavior of the device much more reproducible. And this is important, not only for the user who has a good experience that this device really behaves as prescribed, but it's even more important for the vendor dealing with the support for millions of customers. So, for any device that is being deployed in very large volume, reproducibility of its behavior is very important.

Hal:

And immediately things like high performance compute, large grids, even, I would say, some of the emerging market attention being paid toward cloud computing, start to look at how do we rapidly build out – and I would say, repeatedly build out – instances that are pretty much the

same, but with some minor personalization or minor variation among them.

Olaf:

Exactly. This is precisely the point we try to address. So, we try to rethink the way software is deployed in the data center, and try to take advantage of this firmware model that we use on consumer devices to make this more reproducible, easier to deploy in large number of instances. However, if you look at the data center server, there are still substantial differences between the way we use consumer devices and between the way we use data center servers.

Hal:

And I guess the issue there comes down to exactly how you configure them, and more importantly, I would say, configure them over time. And again, I'm struck by your emphasis on the firmware model, because you can do any number of upgrades to your consumer electronics devices, and only occasionally do you have to download new firmware, and you do download new firmware, most of the time, everything else you have seems to work from that point on, as opposed to having go do a full reinstall or a full reload. So, how does this play out? If I want to go build out a data center with a thousand or ten thousand machines in it, how is Live* going to help me?

Olaf:

The key idea behind Live* is to provide the same thing that we have on consumer device, that we take images with large software components already preinstalled and ready to run – for example, we can have an image with an operating system, an image with a database, or an image with an app. server. And to provide a Lego-like infrastructure so that you can mix and match these images and build more complex systems out of this. And the reason why this modularity is really critical is that, unlike in the consumer electronics space, software in a data center comes very likely from a different vendor than the vendor who provides your hardware; and it also comes in multiple modules that come, usually, from different vendors. It's very likely that your operating system will come from one vendor, database from another vendor and you can use Open Source or some other vendor for an app. server. And what this means is that, unlike on the consumer device, where everything comes, typically, from one vendor, there's Apple providing you an iPhone and firmware for an iPhone, Motorola providing cell phone and the firmware for the cell phone, we try to create much richer abstractions where you can mix and match products from different vendors, and at the same time, have a guarantee that this environment is reproducible and really better than what it was before, when you installed patches and these patches were breaking some stuff that you had already installed before.

Hal:

And I think that more and more, we're seeing that this stack of software that runs the enterprise data center – more so than coming from multiple vendors, is coming from multiple points of presence. It's as much a combination and confluence of a variety of Open Source software projects as it is just a mix and match of your favorite database, you know, web tier software and operating system. And you may have eight or ten or 12 key components that go now into building a web-facing infrastructure. And just the mechanics of producing that, of getting all the packages installed, getting all the interactions nailed down, I think it tends to leave a fairly broad trail of system administration to breed behind. And you know, anything we can do to regularize that and to eliminate the complexity, I think, is a good thing.

Olaf:

Yes, here is a very important point, that in the environment where you use multiple components

developed by multiple vendors or multiple Open Source teams, the end user struggles to get all this working together, to get the right versions of these modules so that the entire stack really works. And as much as we value, you know, being able to select different versions and customize and so on, if you start being more pragmatic and just operate a large grid and swapping between versions of software, you like to take advantage of competence, either of your Open Source community competence, or competence of some service or integration partners to provide a set of packages that simply work together out of the box, and they are compatible. And in the enterprise world, you want to basically have this in some sense certified, so that somebody will provide third party support for that kind of environment.

Hal:

Okay. So, I want to dig a little bit into the mechanics of this because when you say firmware, I think combination of things which are read-only, and yet, I want to be able to do package installation, which is going to involve modifying things. What's the secret sauce here in Live* – how do you make an underlying operating system look like firmware, and how do you make the packages running on top of it look like personalization? What's the level of abstraction you're introducing?

Olaf:

Okay, so we have combination of two different mechanisms that cope with unstructured data and with structured data. So, if we look at any software installation, if we take two machines where we installed basically the same set of packages, they basically differ only with some personalization settings. And that's what we call structured data. Most of other files in the file system with installed software will be probably identical on both machines. At the same time, all the applications expect software to be located – I mean, especially things like configuration files, libraries or common executables – to be located in well-known places in the file system. So, we expect configuration files in /etc we expect libraries to be in, say, /libar or /lib64; we expect all the common binaries to be in /usr/bin. So, if we try to provide software in a modular way that, for example, that the operating system comes in one module, database in another module, but they both provide libraries and configuration files and common executables, we take advantage of something called File System Unification – for people familiar with BSD, this is typically called Union Mount in that world, in Linux it's called Union File System, Sun once had Translucent File System – basically, technology that allows us to overlap name spaces of multiple file systems to create an abstraction of the file system that has, for example, /lib64 that contains libraries provided by multiple images. So, in this virtualized union file system, /lib64 will have libraries from the operating system and the database and app. server. As far as the structured data is concerned, it's usually a very small fraction of the data in the whole file system, but the problem we see today is that configuration files have variety of syntax flavors and variety of ways they are being used. And what we do in Live* is we combine all the personalization settings – basically, all the inputs that, traditionally, system administrator provides to installation programs, to a common data structure which contains a list of key value pairs with the questions and answers. And at the boot time, we regenerate the configuration files in yet another in-memory image, and we combine both the software images – these kinds of firmware images, which remain immutable with the configuration files created in yet another image. And with the writeable per-instance file system that allows us to create an impression that the whole file system is writeable, while all the firmware images remain immutable.

Hal:

So, what you've done here is you've taken system administration as the sum of the history of all the individual changes that have been made, and really turned it into a set of deployment descriptors that can then be applied to this disk image or set of, you know, fixed packages,

immutable packages, as you call them. And the net sum, when you take the union file system, so the file system name space unification, tied together with this abstraction of configuration data into structured and unstructured data, you end up with a really nice way of programmatically, I think, describing configurations. That's the magic here, is getting the configurations right.

Olaf:

Yes. So, effectively, we pushed back software installation back to the software factory, and we ran the installer in the software factory to create these precooked, ready-to-run images, and we create the infrastructure that allows us to apply all the personalization settings to the system at the boot time.

Hal:

So, who's your target audience? Is this software developers, system administrators, people who are in the middle, who are sort of scriptwriters for people who, historically, would have played it with tcl and tk and are now Perl and Python hackers? Who do you want to be thinking about being the core group of users and developers around Live *?

Olaf:

So, you can look at Live* both as an end user or deployment tool. And in this case, our main target right now would be people who deal with large installations of similar machines. So, for example, HPC clusters, or you know, larger-scale enterprise cloud computing, where you need to deploy many machines and you try to do it in a very similar and reproducible way. And to some extent, this overlaps those people who used to write Perl or Python scripts to automate these deployments. But, the other way of looking at this is to look at the entire software distribution business at the vendors who are right now puzzled what to do – Shall we distribute our software as virtual appliances, or live CDs, or something like this? And we provide, to some extent, a new way of building software and delivering software to the end user, which makes this easier to use, more reproducible and also easier to debug for the vendors.

Hal:

Great, and like all good change, I think you've talked to both the developer side of this and well as the deployer side; and being a system administrator by trade, anything that makes the life of deployers easier is bound to, over time, reduce cost in a data center. So, Olaf, where do people go to find more information and to go give Project Live* a try?

Olaf:

I would encourage everybody to look at the Sun Labs webpage at Sun; it's <http://research.sun.com>. One of the links on the main page points to all the projects in the Sun Labs. Among those projects, you can also look at the related projects, as Project Caroline, and so on, there.

Hal:

Okay. And we'll have the source code and trial version available on our website, as well?

Olaf:

Well, our Open Sourcing effort is underway, so it's not yet there. It will be there in the near future. And we are also looking for good early adopters to work with them on this kind of pilot deployments, where we would like to try this technology and this approach the real world.

Hal:

Great. Well, I'd like to thank you for coming and joining us and talking about Project Live*. Again, you've been listening to Olaf Manczak, who is Senior Staff Engineer and principal investigator of Project Live* in Sun Labs. And you have been listening to another episode of Innovating@Sun. I'm your host, Hal Stern.

Announcer: You've been listening to Innovating@Sun. Join us next time for the latest in innovation from Sun Microsystems. Only on the Sun Microsystems Podcast Network.