Issue November 2012 | presented by jaxenter www.jaxenter.com

jaxmagazine

#23

JavaFX Revitalised –
Clocking hours with the Canvas API

Actors Studio –
Using Akkas Asynchronously

Master Craftsman –
Learning the Ways of Software Craftsmanship

Tomcat Extended –
Talking TomEE with David Blevins

Embracing the new guard

We're beginning to sound like a broken record, but it's become apparent over the past year or so that Java innovation is being driven by the supporting cast.

Whether that's the plethora of JVM languages such as Groovy or Scala, or the number of frameworks cropping up, it's an exciting time to be in the industry as we all try to solve the challenges that face us.

For this month's issue of JAX Magazine, we wanted to reflect as much as that thriving cross-section as possible. JavaFX 1.x was an unmitigated disaster, but JavaFX 2.x looks to have amended those problems in a fresh start. JavaFX enthusiast Carl Dea shows us its potential using the Canvas API to create a Tron Legacy-style clock.

Scala has enjoyed a fruitful 2012, mainly due to the growth of the Typesafe Stack. Jamie Allen shows us how to get the most out of Akka, using actors asynchronously.

In this new developer age, coding is not enough. You have to be able to master other things too – Uwe Friedrichsen is our sensei in helping us understand what software craftmanship is and how we can become a true craftsman. Meanwhile at Eclipse, they're putting a lot of their efforts into machine-to-machine projects and see it as big part of our future. Benjamin Cabé tells us more about Mihini.

We also chat to TomEE lead David Blevins about the Web Profile version of Tomcat pushing the boundaries. In the State of Play, we attempt to make sense of the array of cloud infrastructure projects out there – who will reign supreme out of CloudStack or OpenStack? Or is it even that simple?

Finally, the JAX Magazine team stretched their legs and we're on the scene to cover JAX London 2012 – suffice to say, it was a tremendous event.

Enjoy the issue and embrace the new guard!

Chris Mayer, Editor

Index

HOT

Hadoop goes real-time

Finally the big data technology looks to be overcoming the hype and actually giving data crunching at the speed enterprises expect. Two Hadoop vendors, Cloudera and MapR are putting their faith in new open source projects, in Impala and Dremel respectively. Although both projects differ in their *raison d'etre* (Dremel is more analytics-based, whilst Impala focuses on querying), it's now become crystal clear that Hadoop's groundwork is solid enough to pursue a world beyond batch. (*Chris Mayer*)

Waratek

We had the pleasure of talking to these guys at JAX London and we're impressed with their product that claims to truly take Java to the cloud. Through a cloud-tuned JVM (which promises to be able to work with SaaS, IaaS or PaaS), Waratek claim to make the troubled ground of multi-tenancy, scalability and elasticity a cinch for developers. It's a bold move but could be the one for you. (*Chris Mayer*)

Java 7

It might be overdue, but we're seeing some signs that Java 7 is finally getting the adoption rate it deserves. Case in point - cloud platforms. The team behind hugely popular Google App Engine has already begun rolling out some language features and full Java 7 support is not long off. Despite the likes of Cloud Foundry and Jelastic already offering it, having Google pledge their support might see droves of developers finally get over their Java 6 hangover. (*Chris Mayer*)

NOT

Google Dart

Despite finally cracking the TIOBE top 50, Google's effort to usurp JavaScript just doesn't convince us, especially with Microsoft, Apple and Mozilla all giving the browser thumbs-down. With those three all refraining, you have to wonder where the momentum is going to come from. Thoughtworks also rated it as 'Hold' in their October Technology Radar, suggesting trepidation. Is this project doomed or can Google turn the tide for their fledgling language? (*Chris Mayer*)

Windows 8/RT

With the dual release of Windows 8 and the Surface tablet taking place last month, it's difficult to escape from talk of Microsoft's new platforms (especially with a rumoured $1.5 billion marketing budget). Yet JAX Magazine's advice is to stay as far away as possible – not because it's bad per se, but because the new Metro interface can't even run Java applications. It's so locked-down, you'd almost think it was some sort of Apple platform. (*Elliot Bentley*)

JAX London 2012

Lambdas, data crunching and hacks, oh my!

For the fifth time, the JAX London conference series landed at Park Plaza Victoria promising to explore some of the biggest and most talked about Java innovations, in sessions and keynotes. Debuting alongside the event was Big Data Con London, a data-focused spinoff discussing analytics, the hottest databases and use cases. The JAX Magazine team were of course there to cover it all...

by Chris Mayer

The Road to Lambda: Kicking things off on Tuesday 16th October was Oracle's Java Language Architect Brian Goetz, delivering a keynote detailing the progress of Java 8's most exciting feature – closures in Project Lambda.

The Sunday before, Goetz teamed up with the London Java Community in holding an exclusive Java 8 Lambda Hackday, giving 50 developers the chance to 'lambda-fy' an existing application with the Head of the Lambdas Expert Group and he began by thanking them for the opportunity.

In a refreshingly honest yet informative keynote, Goetz explained the difficulties in balancing what the community want against compatibility, and staying true to what he called keeping the language close to – "the spirit of Java." He even acknowledged that Java is "the last holdout" in a world where every language can already boast closures. But despite all that he also exclaimed "it's been a long journey, and it's nice to be nearly at the end of that".

To conclude, the Java architect accepted that they were "catching up with the competition" but this was a push "towards functional programming, without using the F word."

Hadoop's origins and bright future

Making a pitstop in a quick European trip, the creator of Hadoop and Cloudera's "James Bond" Doug Cutting gave the second keynote of the first day, charting the progress of the data crunching platform and even offered a vision into its future.

Cutting regaled the audience with tales of Hadoop's origins at Yahoo, before it swiftly found itself in the development environments at Facebook and Twitter. Then onto enterprise domination. He told the crowd that he believed Hadoop was strong enough and robust enough to be adopted by anyone now, but Cutting was keen to emphasise the importance of the supporting cast that appear in the Bigtop distribution.

He explained Hadoop's real value didn't come from the components alone, of MapReduce and HDFS, but the two in unison, adding: "The interesting part is the reliability – that's why people use this. That's not easy to get, but once it's there, the framework acts as a basis".

Cutting finished by discussing what he saw as the "Holy Grail" for data – and that's Google's recent paper Spanner. "It's an impressive piece of work and gives me great optimism looking at the framework that it has legs".

Not long after, Cloudera released Impala, heavily inspired by Google's F1 system. It's clear to us that Google is the light that shines the way – and now we're starting to see Hadoop go beyond its batch heritage towards real-time. An exciting time indeed.

Java developers need to be more like Sarah Connor

Blockbuster references were heavy this year at JAX London, especially in Tuesday's Community Night keynote. LJC co-lead Martijn Verburg, alongside his JClarity colleague Kirk Pepperdine, entertained a packed house in the heavily Terminator-focused "Java and the machine", despite the latter having never seen a Terminator film. The duo warned developers that they need to be equipped with the right weapons in this multicore age, and let software and hardware work in harmony.

Pepperdine and Verburg riffed off each other well, as the beer flowed. The Java performance expert heavily played upon his own phrase "It's about the hardware stupid", whilst Verburg (donning his beanie and shades) assured us all that there was no need to fret, if you just learned some simple laws from Moore, Amdahl and Little. He also used a useful multi-threading analogy too, comparing it to managing children in a play area. "You think one is easy, try, 2,4,8,16" which drew a big laugh from the crowd.

Collaboration is king

Suitably refreshed for Day Two, IBM's Java evangelist Steve Poole set a trend for Wednesday, using his keynote to encourage developers to get involved with OpenJDK. With a heavy LJC presence, Poole noted their work in breaking down the boundaries between developers and suits, before outlining Java history, today's challenges and what we can do about it. Poole repeatedly told the audience that "we" (the developers) could take Java in which direction they wanted if they shouted loud enough and tested enough. He urged the audience to reflect on what makes newer languages like node.js and Clojure so attractive, and use their parts as inspiration for Java. "When you say, 'I want to learn Java, but...', tell us what that but is," he said. Feedback is essential according to Poole, a big driver behind OpenJDK "Writing blogs, reporting bugs, writing code – if you've got the time and resources to do it, then go do it!" he concluded.

The final keynote was delivered by the man who coined the term Devops and he was keen to preach sharing too. Patrick Debois provided a comprehensive overview of the environment changing mantra that is sweeping the enterprise world, and how really simple fundamentals often get lost in translation. He added: "You're thinking this is so obvious but it's not. It's like telling your children to brush their teeth – they know they should do it, but you have to keep telling them."

Debois told the crowd how to begin thinking about merging the world of development and operations and how much you need to consider, especially with sharing. Whether it's a codebase, a bug tracker or even shared success and failure – it's all part of the Devops mantra.

www.jaxlondon.com

Cloud infrastructure projects

The State of Play

With OpenStack, CloudStack and Eucalyptus gaining strength, we look into this 'war' of the open source cloud and attempt to predict the outcome

by Chris Mayer

It barely seems like a day goes by without another announcement from a cloud infrastructure project. Whether it's a new vendor signing up or the less likely news detailing progress of the project, it's hard to really escape chatter surrounding the project.

You have to feel sorry for the prospective buyer, keeping tabs on how these projects are developing. Wading through the marketing material is no easy process, let alone working out which cloud orchestrator is the best fit for your business.

The main contenders appear to be CloudStack (incubating at Apache, but has finally seen some code emerge) and the vendor melting pot IaaS OpenStack, which has recently been handed to the community controlling the project. The two are ideally placed to gain greater adoption in the coming months. With CloudStack at Apache, they can expect the tender loving care given to all their other projects. The calibre of companies investing in CloudStack by default makes them the top dog though, due to the juggernauts all signing up. But is there another open cloud project coming up on the outside? VP of Community at Zenoss, Floyd Strimling thinks there's plenty left still to run in this race.

What is an open cloud anyway?

It seems logical to start with a definition – just what exactly is an open cloud? This is all down to interpretation of course, but the fundamental aspect should be that the finished cloud environment is formed of open source technologies. Whether this is then turned into a commercial offering is irrelevant (and certainly naïve) since normally we expect open source projects to branch into commercial operations. Some might choose to enforce stricter rules upon that, but as long as open source is the very core, there shouldn't be any qualms.

Strimling agrees saying "you have to make room for a commercial open source in the equation" He continued: "It's the idea of being able to take components, interchange them, switch it and move them into different directions and be able to have no vendor lock-in because it's so open. And because you're able to get really deep into the code you're understand what's happening, rather than being reliant on another entity."

The push

So why the current move away from walled offerings towards something far less restrictive? Strimling believes there's a number of reasons why software and hardware vendors are singing from the same hymnsheet, the most obvious being *vendor lock-in*.

"If you want to go with CloudStack, OpenStack or Eucalyptus, there's so many open options. We finally have the flexibility to choose what meets your business calls, not what you're being forced to use."

There's a general misconception amongst companies that moving towards something more open is more cost-effective, when this isn't yet true. What it does offer is *greater agility* right at the heart of development, to allow the changes you want, which in turn might seem like you're getting better value.

Strimling thinks there's also a *general curiosity* to explore the possibilities, with companies beginning to wonder whether they can build something new and innovative.

"The first quarter of the race has all been public. I think that there's enough for people to finally understand that with a private cloud, you achieve agility and you can also achieve some of the same types of flexibility that you can get out of the public cloud, but in the confines of your datacenter," he explained.

Common concerns

Naturally, there continues to be reasons to steer clear of open source clouds, as there would be with any emerging technology or field.

Strimling cites three in particular as common concerns: *maturity, lack of support and security*. Whilst the first two will probably take care of themselves as more vendors jump onboard, enterprises continue to be wary of pushing their data out into the unknown, so to speak.

"I think that security is starting to take care of itself in that that some of the scares around it were basically the fears of running unknown. I almost claim it's like a fear of flying. The fear of flying is really not being in the air, it's the fear of not being in control, turning it over to somebody else."

The term 'hybrid cloud' however should approached with legitimate caution. From a marketing perspective, it might sound straight forward – public cloud connected to private cloud and voila, hybrid cloud. From a logistics perspective, it isn't this simple. Strimling believes the marketing term is rhetoric for being able switch between specific requirements and being on and off-premise.

"Right now, a lot of the applications we depend on for everything that we do are monolithic and are not cloud-friendly, no matter what the vendors say. SAP is a wonderful package of tools and things you need to run your business, but to think you could actually deploy that in a hybrid cloud environment, no way. It's not ready yet, the app vendors have to catch up as well."

CloudStack vs OpenStack

As previously mentioned, these two that seem to have the biggest buzz surrounding them, likely fuelled through the communities behind them. They share common ground, in the sense they are in their infancy yet both have key differences.

Apache CloudStack is undergoing heavy tinkering to make sure it's completely compliant with the Apache Software Foundation's core rules and beliefs. After all, CloudStack initially arrived in 2010, created by Cloud.com and Citrix. The solution needs to shirk this tag quickly and prove its open source credentials, which in our opinion it has in the recent 4.0.0 CloudStack release – the first with the Apache branding.

It does have a head-start on OpenStack though. What started out as an exploratory project by NASA and Rackspace has now blossomed into a fully fledged community that has design summits every six months. With every release (Grizzly next on the list), we see new components bolted onto the project to add new functionalities. With the two founders stepping aside and handing the keys to the collective, OpenStack should pick up momentum quickly.

Of the vendors putting their faith into the project, Red Hat have certainly been the most active – development-wise and vocally. Strimling believes they have the biggest shot of success, indicating that the huge open source company "have the greatest chance of all because they have the distribution and channels to actually use it."

OpenStack's biggest problem is the maturity of the project. Some parts are tried and tested (Strimling describes OpenStack Storage, formerly Swift, as wonderful) yet others are in infancy and this imbalance doesn't sit well with the budding enterprise looking to pick it up.

With more than 150 parties with a vested interest, OpenStack does find itself on shaky ground. They might be on the right track with innovation but what happens when one company hits upon the pot of gold. Red Hat CTO Brian Stevens recently told our sister publication jaxenter.com that "the future is already written" for OpenStack yet Strimling doesn't think it's as clear cut as this

"I don't believe the winner has been crowned" he told us, adding that "there'd have to be a clear path to commercialisation which for right now, there is no clear path to commercialisation with OpenStack. There's a clear path to commercialisation on the CloudStack front and the Eucalyptus front."

"I think it's way too early to crown OpenStack as the winner and I wonder what if this wonderful OpenStack project, what's going to happen to it when it starts making money, and it's not you?"

At the end of the day, Red Hat have to make OpenStack a commercial success and are well within their rights for doing so. But for both to permeate the business circles they desire, the true test comes from targeting their products away from project leads and go mainstream. Then we might get an indicator of where things are really heading.

The horse on the outside – Eucalyptus

One product that doesn't gain nearly as much attention is Eucalyptus. The company fronted by the man who was MySQL CEO Marten Mickos has gone untracked despite having a fairly solid offering that enterprises should be checking out.

Strimling puts this down to "missteps" when they launched, and people misunderstanding what they were about. Some even thought they were operating out of Amazon, due to the rock solid AWS hook-up. He told us: "I think right now people don't know enough about Eucalyptus, they don't know enough about its differentiators. They're getting lost in this OpenStack/CloudStack and it's not that model at all. "

"It's not governed by a foundation, so the software is all being controlled by Eucalyptus. It's in the same vein as MySQL was created and done, so you have a really free and open thing but support is coming from one company."

They may be lacking in the media front and star power of the other two, but they deserve your attention.

The winner hasn't been crowned

It's a fairly stock answer but the truth, it's too early to call. CloudStack isn't out of the Apache Incubator yet, OpenStack has some choppy waters to navigate and Eucalyptus need to separate themselves from the pack.

That's of course without mentioning the real kingmakers in all this – VMware (who were recently let into the OpenStack Foundation), Google (who could crash the party) and of course the cloud heavyweight incumbant Amazon. Two of the three have AWS functionality, one has decided to strike it alone. That might prove crucial. It ain't over by a long way yet regardless.

http://mediasrc.zenoss.com/documents/The_State_of_Open_Cloud_2012_infographic.pdf

Tomcat Extended

TomEE 1.5

We chatted to David Blevins about the project that takes Tomcat one step further – for this generation's applications.

JAX Magazine: Best place to start I guess, what is TomEE, what was the thinking behind starting it and what problems does it set out to solve?
Blevins: TomEE is the Java EE version of Tomcat. The idea was there and many people have wanted to see Tomcat go beyond just servlets for quite some time. If you trace the origins of TomEE, you can go as back as far as the early 2000s with the OpenEJB Tomcat stack which is also where the EJBs in WARs feature of Java EE 6 originates. The EJB 3.1 Embedded EJB Container API comes from there as well. The thing that finally made TomEE possible, though, was the addition of the Web Profile to Java EE 6.

JM: So was it feasible or way off at that point?
Blevins: Creating TomEE was incredibly difficult. Having implemented Full Profile EE servers for over 10 years, I knew it was going to be hard even with some of the legacy parts of Java EE cut out and not in the Web Profile. Once we dropped the basic parts, we were still only 40 % there. It took almost a year to fill those gaps. It doesn't take a lot of code, but it takes the right code. I think part of the real value of TomEE is that we have put so much work into the completeness of the integration and have done it in an incredibly simple and direct way.

JM: Can you outline the main parts that go into TomEE?
Blevins: TomEE includes all of the Web Profile technologies, all of which are very relevant. Just to list them off, beyond Servlets and JSP we have JSF, CDI, JPA, JTA, EJB Lite and Bean Validation. That's the core. They're all incredibly important. I think actually of all the technologies, Bean Validation tends to get the least buzz [about it], but it's perhaps the most relevant to every single one of those specifications. If you're using JPA, you need Bean Validation.

JM: So it's something developers might take for granted I guess?
Blevins: Yes, definitely. It's not a large API, but it's definitely important. Validation is as important to data as unit testing is to code. With all the focus there is on testing, there really needs to be an equal focus on valid data.

Portrait

David Blevins is project lead for TomEE, co-founder of the Apache OpenEJB project, a founder of Apache Geronimo, and contributor to many other Open Source Java EE related projects for over 10 years. David was an active member of the EJB 3.0 (JSR 220), EJB 3.1 (JSR 318) and Java EE 6 (JSR 316) Expert Groups, and contributing author to Component-Based Software Engineering: Putting the Pieces Together from Addison Wesley. He can be found speaking on these topics at JavaOne, JAXconf ApacheCon, and O'Reilly Open Source Convention.

JM: Was CDI an absolute must for your team for the main release?
Blevins: We all saw it as critical. CDI is a very compelling technology. It's essentially the new integration layer for Java EE and in a lot of ways it obsoletes EJB. At the Java EE level, we're moving more and more onto CDI.

> *"Validation is as important to data as unit testing is to code."*

JM: There's obviously a heavy reliance on Tomcat - taking onboard its good points and moving beyond it. How close are the ties between TomEE and other projects?
Blevins: All the projects are at Apache and the ecosystem there is very close. We all collaborate quite a bit. When we put out a TomEE announcement, we have people from Tomcat, MyFaces and OpenWebBeans all chipping in and helping. It's definitely a good community. In technical terms, though, the way Tomcat has been written means that we need almost nothing to be added or changed in Tomcat itself to make TomEE possible, which is a really good testament to that project.

JM: So it's refinement then?
Blevins: I'd describe TomEE as an extension of Tomcat. The Tomcat architecture is pretty amazing when you think how old it is, it was really ahead of its time. It is very much an event-based architecture and it was through that event-based architecture that we were able to make these great extensions to how things are deployed, started and stopped. We didn't have to change anything architecturally. We just followed the path that was laid out there. Hats off to Tomcat for having that really flexible architecture.

JM: So what does it allow you to do that regular Tomcat doesn't then? What scenarios does thrive in?
Blevins: In terms of what Tomcat offers out of the box, it's about 2/7ths of the Web Profile. So there are a significant number of extra technologies in TomEE. These are the typical technologies that people usually add themselves. I think that TomEE thrives in any scenario that involves Tomcat and the reason is because often you start out with Tomcat installed and then the first thing you do is you go to build it to be more than Tomcat...

JM: And if the extra stuff is already there, there's no need to go hunting.
Blevins: Exactly. From day one there is a better alternative. Adding these things as you go can cause several problems. The first one is it can be a huge waste of time. You start with a basic Servlet app and now you want to persist some data to a database. So now you need to add a JPA provider. That one is easy, but then every time you learn about a new bit of technology, it becomes harder and harder. Say you want to add CDI, but you don't know much about it so you have to both learn CDI and how to integrate CDI into Tomcat and with all your other libraries at the same time. So you're coupling learning about a technology with integrating that technology. Having to integrate a technology you know nothing about is a terrible position to be in. You might get something working, but you don't really know if that's the way it should work and every time you run into a problem you don't know if the problem comes from not integrating it properly or if it is an actual bug or if you're misunderstanding completely and expecting the technology to do something it isn't supposed to do. It's an incredibly frustrating position to be in. We then repeat this on every new technology we want to add to our application.

We're all mature enough to know that we don't "just use servlets", as if the only useful Java APIs have already been created and everything that came after is all stuff no one ever uses. As if the average war file doesn't have a ton of additional libraries in it. These days even new war files are packed with a ton of libraries we know we'll need even before we've written a line of real code.

> *"Hats off to Tomcat for having that really flexible architecture."*

Second, just because you add a library to a webapp and can get some use of it, doesn't mean it is fully integrated. For example you can't drop CXF into Tomcat and expect Web Services security to work with the Servlet security. You can't drop OpenJPA into Tomcat and expect JTA-managed EntityManagers to work. CDI, for example, has a strong integration focus, but even if you drop OpenWebBeans into Tomcat you've still only got 60 % of what that spec has to offer. CDI injection is not going to work on your Web Service, because they're different technologies. So again, you're back to Googling, writing integration code and solving this problem as if no one's ever solved it before. As the number of useful Java technologies the industry has to offer grows, it just becomes harder and harder to do it yourself and keep everything consistent.

Then there are performance considerations to be had. APIs do a tremendous amount of scanning do these days, which involves reading every single class file in an application looking for annotation declared components. Tomcat will scan your entire Web app for servlets, listeners and filters. Then if you add MyFaces it will scan your entire Web app for manage beans. Now it's scanned twice. Then you throw OpenWebBeans in there and now it's going to get scanned a third time. You throw OpenJPA in there and it's going to scan your webapp a fourth time looking for entity beans. Throw CXF in there and now CXF is going to scan your entire Web app for Web Services. It's incredibly wasteful, it's not quick. Not only has the number of scans gone up but the amount of jars that have to be scanned has gone up. OpenWebBeans is going to scan MyFaces, OpenJPA and CXF. And CXF is going to scan OpenJPA, MyFaces and OpenWebBeans (laughs).

JM: Wow. That's a bit time consuming.
Blevins: It's not entirely efficient, no.

JM: I think that's fair to say.
Blevins: These are some of the things we take care of in TomEE. We scan once to the best of our ability, we share that information with MyFaces, OpenWebBeans and Tomcat. We try to cut the redundant scanning down much as possible. It's not perfect and we're still going down that path, but we're very far down it. It's certainly more than you'd get by just throwing those individual specifications on Tomcat.

Another aspect is the level of testing we do as we make improvements like these. I'm not sure how many tests other people have for their Tomcat stacks, but I can say from a TomEE perspective, we run hours and hours on TomEE every day because of the very large compliance test suite that is the TCK, which all certified implementations have to pass. If you do it in linear time, it's days of tests. If you break it up and parallelise it like we do, it's a couple of hours of tests but still a couple hundred hours of machine time. That's a significant level of protection you're getting knowing that everything you have in there is complete and passes these set of tests. And you can get that in 27mb.

JM: Yep, that is impressive.
Blevins: It's not a large tradeoff either. The really critical thing we're trying to do with TomEE is to not subtract from the Tomcat experience, only add. It's a difficult challenge and we work extremely hard at it. We never claim perfection so if you find something that ruined your Tomcat experience let us know and we will fix it.

> *"The really critical thing we're trying to do with TomEE is to not subtract from the Tomcat experience, only add."*

JM: I remember seeing on the original 1.0 release, TomEE boasted that it was 300% faster. How was that achieved?
Blevins: A lot of that was done through reduced scanning. So, Confluence is about 200MB and 199 JARs. When we boosted the performance, we didn't get it tuning things using the approach where we magically know what to skip. That would be unfair. When someone in the wild throws a 200MB file at you, you don't have the luxury of knowing which parts of the Web app are important, which ones aren't and being able to tune accordingly. We just really really revamped all the scanning code. We got the scan time alone down from about 8 seconds to 2 seconds which really cuts down on the overall deploy time. And then there was another round of further tuning. It was just a lot of cranking on the bolts and tightening of screws to get it into production shape for a final release.

JM: You passed the Oracle TCK test, which is quite the feat. How long did that take for a start?
Blevins: Yeah. We started in about February 2011 with our first full run of the TCK. We were probably about 20 % compliant then.

> *"We try to cut the redundant scanning down much as possible. It's not perfect and we're still going down that path, but we're very far down it."*

JM: Not a bad starting point there then?
Blevins: We had a good head start with a lot of the parts there already like EJB, JPA, JSF and obviously Servlets, JSP. But CDI was in constant flux and in general there are a lot of things you discover are missing features and hold you back from completing large sections of the TCK. It's all about filling in those gaps as quickly as possible. From there, we were able to reach full compliance by October. But it was close, we got our first pass at the end of September. Maybe a week and a half before JavaOne. We cut it so close.

JM: Perfect timing though
Blevins: We had our minds made up to make the JavaOne date. It was an incredibly challenging goal for the community and we really came together and made it happen.

JM: I can imagine there were some late nights around that period.
Blevins: Many of us had really late nights and really early mornings. I know myself, I was sleeping four hours a night for several weeks. It didn't stop until after JavaOne because once you pass the test, you have file your paperwork, you've got to make an announcement and get a release out the door and do it while preparing for JavaOne. That particular year, I spoke at six different talks, so I had a lot of material to produce. I don't think I really saw anybody at JavaOne (laughs). After that, I collapsed.

JM: Moving on to Java EE - what are your thoughts about recent developments? Is it heading the right way?
Blevins: Absolutely. I think that when Java EE 7 started out, it was assumed we would want to do some cloud-based things.

JM: Because of the move toward standardisation...
Blevins: Initially it was an obvious goal. But as we developed and went further down that path, it became clear that we weren't ready to do that yet. The fact that there was enough self-awareness shows it's an astonishing victory for standards because we all know what happens when things are standardised too prematurely with a non-proven API. We end up

having an API that is a false start and we have to figure how to bury it and supersede it with something. It's not a good position to be in and it's not what's standards are supposed to do. The fact that we could perceive that this was something that wasn't ready to be standardised, I think is exactly the kind of things that standard bodies should be doing. I really compliment the group for doing that.

That said, there are a number of really cool things being done in the various Expert Groups and being talked about in Java EE 7. JMS 2.0 is going to be a really great part of Java EE 7 as is JAX-RS 2.0. Then there are things coming from the TomEE side as well. I had hoped that TomEE would be an important contributor to the specifications in the same way OpenEJB was to the EJB specifications. It's looking like we're going to be able to add some neat things, or at least inspire some changes.

We have a proposal for revamping the Connector Architecture and MDB relationship to handle modern APIs. Something like JAX-RS could have been done via the Connector Architecture with the right tweaks to that model. It sounds a little bit boring on the surface, but imagine being able to create an expressive API modelled like JAX-RS but for consuming email or exposing an SSH interface or accepting Thrift requests and dropping that into any certified Java EE server and having the components that accept those requests just as integrated into the platform as Servlets, EJBs or CDI beans are.

Another one of them is this concept of 'meta-annotations' we introduced it at JAX London 2011. It's a concept of broadening annotation reuse consistently to the entire Java EE platform. These concepts started popping up in various Java EE 6 specifications. Bean Validation which allows for reuse of validation annotations on other validation annotations. So say you add @NotNull on a bean validation annotation, the new annotation implies @NotNull plus the additional constraints it adds.

With CDI, there's a similar concept called Stereotypes. If say @Red is an annotation annotated with @Stereotype, @Named and @RequestScoped or any other CDI aspect. Anywhere you use @Red it implies that component is @RequestScoped and @Named. Now instead of using all those annotations, you just use @Red.

This concept exists, but someone inconsistently in various specifications and there is currently no ability to reuse annotations that are standard annotations such as any of the annotations from the Servlet, JPA, EJB, JSF, JAX-WS or any similar APIs. The value of such reuse is only as powerful as the number of annotations that allow reuse. Currently, that's almost none.

JM: Is that more a matter of time before we see annotation reuse concepts occurring throughout Java EE or does it need a push?

Blevins: No, I think the push has been made now. Aside from JAX London, I've presented afterwards at JavaOne 2011 and 2012. I'll be presenting it again along with the Connector revamp at W-JAX and Devoxx. I've also been working with various people from other app servers on what would be the performance impact of having this annotation reuse enabled all the time. We've already had this implemented in TomEE back in 2011 and we found it to not have a significant performance impact. As we've been working with more people, they've found the same thing. So now it's a question of how we want to do this. The current proposals on the table is that @Stereotype would be what we proposed as @Metatype and would be the annotation that would drive this annotation reuse. Then we would go through and we would change all these Java EE annotations so they could be applied to @Steretype and other annotations.

JM: Good to hear. There was a TomEE 1.5 release recently – was that a case of listening to the community and fixing a few things?

Blevins: We received a flood of feedback from 1.0. So much feedback it was difficult to close the lid and say no more fixes. By the time we did close the lid, we had major new features such as the JAX-RS certification, several of the new tools. The textual description of the 1.5 release notes appears to be very short but if you look at the actual change list, it's difficult to summarise it all. I would say usability, migration issues and rounding off of sharp corners are all very strong aspects of the 1.5.

JM: What are the plans for TomEE moving forward then, if you're allowed to divulge in them?

Blevins: I can speculate of course, but the community sets the direction.

> *"In terms of next major things, obviously Java EE 7 is right around the corner."*

JM: As any good project should.

Blevins: Exactly. I'd personally like to see more of the same things we've been doing. We need to continue to address migration issues. We need to do more along the lines of helping people find the right information to the problems they might encounter. Very basic things like that. There is already a lot of functionality within TomEE which we need to make more accessible and known. Continuing to do that is great. In terms of next major things, obviously Java EE 7 is right around the corner. That's going to be Tomcat 8, Java 7 and will contain a lot of new technologies. That will be an all consuming focus of the project a year from now. Between now and then it's about focusing on basics. I know from experience that you only have very short windows between major spec revisions to really focus on the fundamentals before you have to go back into a very major implementation/integration phase. We need to keep driving down the path we're on, filling out TomEE in terms of monitoring and other types of production concepts and focusing on the fundamentals of the server.

References

[1] http://tomee.apache.org/

Introducing a new framework for developing embedded M2M applications

Eclipse Mihini

In November 2011, the Eclipse Foundation, together with industry leaders Eurotech, IBM and Sierra Wireless announced the creation of an Industry Working Group focusing on M2M technologies. Exactly a year after the launch of this initiative, this article takes a deeper look at the latest project initiated by Sierra Wireless: Mihini.

by Benjamin Cabé

In Eclipse Magazin 5.2012, we already introduced you to Koneki [1], a set of Eclipse plug-ins providing tools for simplifying M2M applications development. You probably remember that one of the main components of Koneki is Lua Development Tools, a complete IDE for developing Lua applications.

During the summer, Sierra Wireless proposed a new project that is bringing the missing yet critical piece to the global architecture of the open source stack the M2M Industry Working Group is working on. The project is called Mihini [2] and its initial source code is currently being contributed.

Mihini is a runtime technology that hides the complexity of the underlying hardware platform to a developer whose primary goal is usually pretty simple: connecting objects (power-meters, cars, medical equipment, …) of the physical world to the Internet, therefore making their data available for consumption by web, mobile, or enterprise applications.

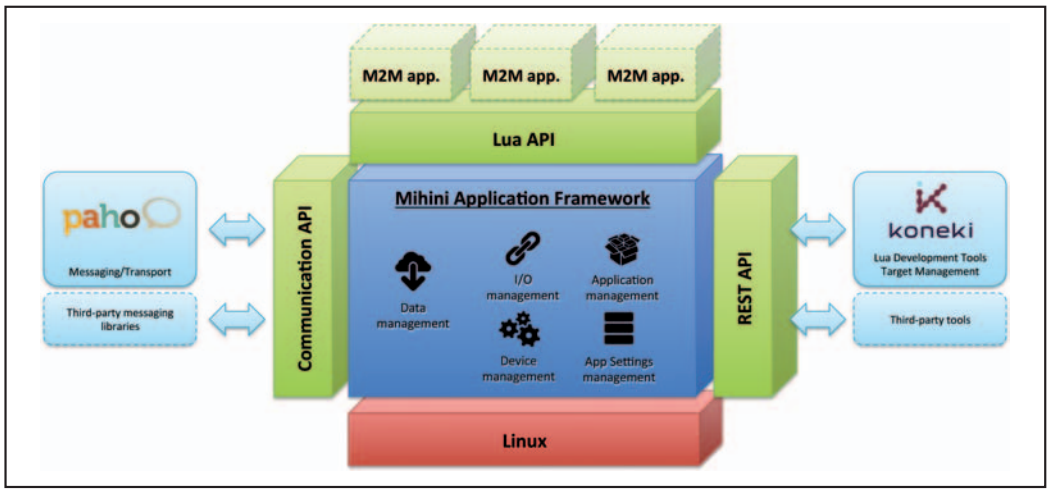


Figure 1: Integration of Mihini with Koneki and Paho

Of course, sending data is not the only aspect of an M2M solution, and things such as being able to react to commands send from a server to the device, being able to remotely update an application, are also of importance.

An overview of Mihini

There is a very good reason why we bootstrapped the M2M Industry Working Group activities and the Koneki project by providing a complete development environment for Lua. Indeed we think Lua is the perfect language for M2M application development, as it is easy to embed in constrained devices, easy to learn, and since it is a scripting language applications are portable by nature.

Also, the Lua virtual machine is very lightweight, written in pure ANSI C, and allows running several applications in a well-defined, sandboxed, environment.

With Mihini we want to provide the minimal set of low-level services enabling reliable operation of M2M applications, and API simplifying access to underlying hardware and easing the exchange of business data with a server:

- **Data Management**. M2M applications that are running on an embedded device are likely bound to physical assets (sensors, actuators) exposing data. Data management API allows to process data in a way that is independent from how it is actually being acquired from these assets, and from how it will eventually be transmitted on the wire. In an ideal world, reacting to the changes of a given input, consolidating its different states/values over a period of time, etc. should be more of a configuration process than writing what will very likely be boilerplate code.
- **Device Management**. Mihini proposes a centralized way for accessing the capabilities of an M2M device as well as managing them remotely. This can be used for example for reporting automatically to a distant server the battery level and signal strength of the M2M gateway, or for forcing a reboot.
- **Network and Connectivity Management**. When an M2M application is deployed on the field, it is to be expected that the wireless network may not always be available, and this for various reasons: poor cellular coverage, planned or unplanned maintenance by the network operator. Even when using a wired (e.g. xDSL) connection, loss of connectivity can happen, and an application may have to transmit critical information even when the main communication channel is down. The Mihini agent allows you to configure the communication channels (bearers) provided by a gateway, and ensures that whenever one of them is down, communication is deferred to another.
- **Application Management**. Mihini proposes an application container that allows to remotely install, update and manage applications on the field. It is primarily aimed at hosting Mihini applications written in Lua, but it can also be used for managing third-party applications written in any language.

The scope of what you can do when writing a Mihini application is not limited to the set of libraries that are initially part of the distribution: the Lua community is very active, and there is a very large ecosystem of open-source libraries available for you to reuse, especially in the embedded domain. Granted that the library you want to use is written in pure Lua, you can include it very easily in your project, by simply copying its Lua source files in your source tree! Some libraries can be a mix of Lua and C code, in which case you will have to compile the native code against your specific target.

Listing 1

```
package = "lua"
version = "5.1"
flags = { ee = true }
description = {
  summary = "Lua 5.1 Execution Environment",
  detailed = [[Lua 5.1 Execution Environment Support]],
  licence = "MIT",
  homepage= "http://www.lua.org/manual/5.1/manual.html"
}
api = {
  file = "api.zip"
}
documentation ={
  dir="docs"
}
```

Supporting Mihini development with Lua Development Tools

The Koneki development team is working hard on the 0.9 version of Lua Development Tools which will be released in December, and most of the new features we have been working on are for enabling Mihini development.

The first major improvement of Koneki LDT is that we now not only allow to develop and debug Lua scripts that are to be run on your development machine, but we also support remote development. We are leveraging Eclipse RSE (Remote System Explorer) to allow the declaration of remote Lua virtual machines. A dedicated *Lua Remote Application* Launch Configuration copies a Lua project and all its dependencies on the distant file system, and executes the application by launching the remote virtual machine via SSH. This feature is available since the milestone 0.9M2, and you can use it with any kind of Lua VM. This new feature is making remote debugging totally transparent, provided that the remote virtual machine is able to reach the debugging server running in the IDE.

We also worked on improving the support of what we call "Execution Environments". An Execution Environment is what allows to enhance the IDE with support of a specific Lua environment, be it a regular Lua virtual machine, your favorite MMORPG, a mobile development toolkit, or obviously Mihini.

An Execution Environment is basically a ZIP file that contains the documentation of the API you can develop against, as well as meta-data that gives some additional hints to the IDE (see **Figure 1** for an example of the manifest file of the Lua 5.1 Execution Environment).

A developer kit for prototyping M2M applications

Developing an end-to-end M2M application is a complex task not only because one needs to master several engineering skills, from embedded to telecommunication protocols to web or mobile application development, but also because actual hardware is needed very early in the design phase of a solution so as one can interact with real assets, and start using actual communication protocols and drivers early on.

Embedded M2M applications are indeed almost always making use of sensors or actuators and not having them at hand leads to situations where it is difficult to validate end-to-end scenarios, from the acquisition of realistic data on sensors, to the communication with a server for displaying the data in e.g. a mobile application. It is usually possible to order a developer kit that features the required sensor from the device manufacturer, but not only is it likely to be costly, it is also usually complex to setup, and may require electronics skills for using it in your design. We want to decouple the software prototyping phase from the actual hardware design, and we think it should be possible to start writing an M2M application even when all the decisions regarding the hardware that will be used have not been taken. Of course it is possible to use simulated data when you start creating your embedded application, but it makes it hard to visualize the actual physical interactions that makes an M2M application unique.

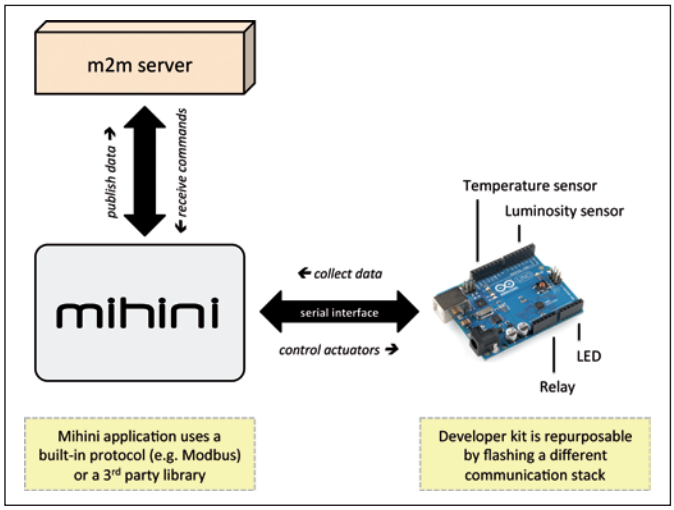


Figure 2: Mihini developer kit

In order to solve this issue, we are prototyping a developer kit based on an Arduino (yes, we do love open hardware!), that includes a handful of sensors (temperature, luminosity, ...) and actuators (LED, relay, ...) and is ready to be connected to a gateway running Mihini. While the hardware used in this kit is nothing very new, the innovation lies in the software that Arduino runs: depending on your use case, you can flash a dedicated program for enabling access to the sensors and control of the actuators using your communication protocol of choice. If you program the Arduino

Listing 2

```
local sched  = require 'sched'
local modbus = require 'modbus'
local mqtt   = require 'mqtt_library'
local utils  = require 'utils'

local function process_modbus ()
  local values =
        modbus_client:readHoldingRegisters(1,0,9)
  for data, address in pairs(modbus_address) do
    local val = utils.convertRegister(values, address)
    mqtt_client:publish(MQTT_DATA_PATH..data,
val)
  end
end

local function main()
  modbus_client = modbus.new(MODBUS_PORT,
                             MODBUS_CONF)
  mqtt_client = mqtt.client.create(MQTT_BROKER,
                                   MQTT_PORT)
  mqtt_client:connect(LOG_NAME)
  mqtt_client:subscribe({MQTT_COMMAND_
PATH.."#"})
  while true do
    sched.wait(1)
    process_modbus()
  end
end

sched.run(main)
sched.loop()
```

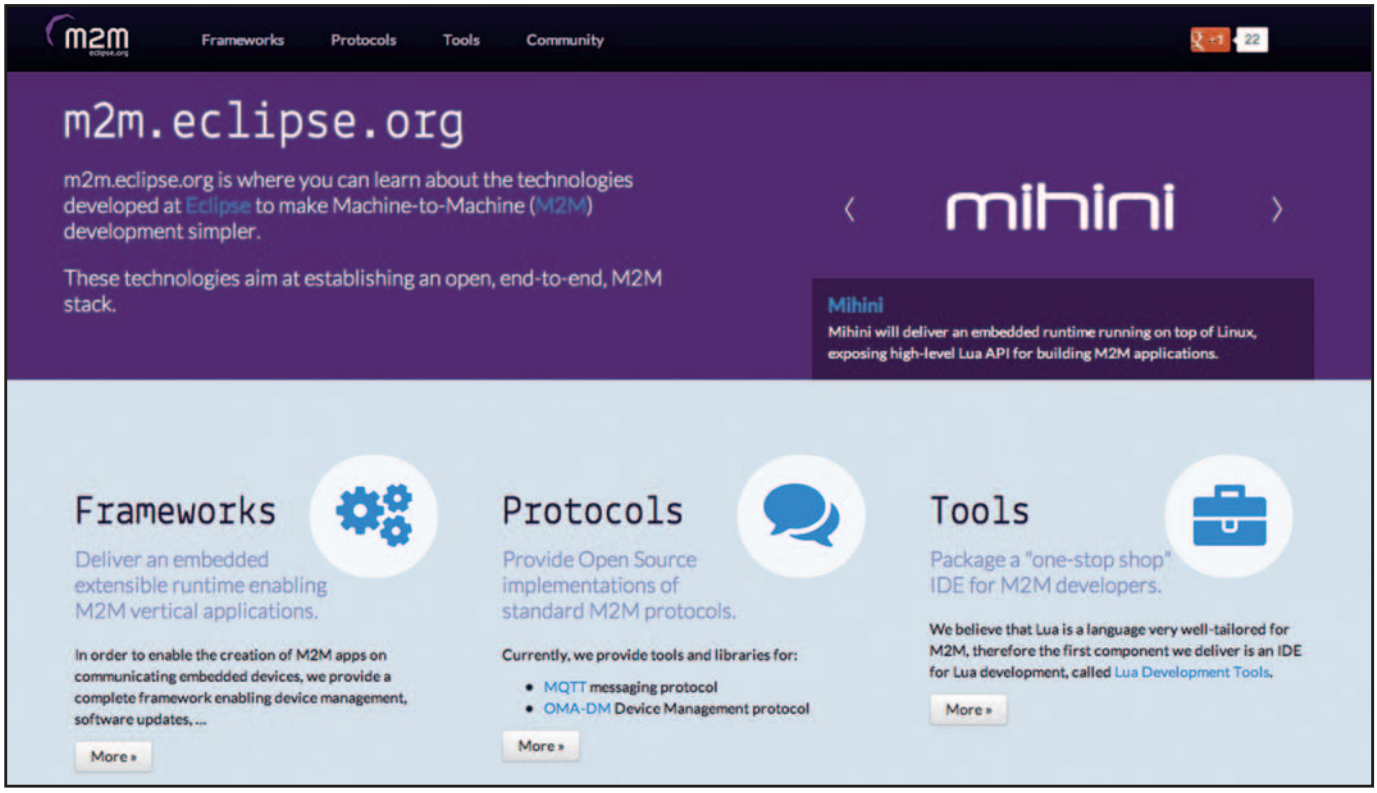


Figure 3: A brand new portal for Eclipse M2M technologies

board with a Modbus stack, you can therefore connect the developer kit to the serial interface of any embedded computer running Mihini, and use the APIs of the framework to control the Modbus registers programmatically. Whenever you are ready to use actual Modbus devices, there should not be much to change in your application, besides maybe the addresses of the Modbus registers. Listing 2 gives you a taste of the Mihini application you could write for reading the Modbus registries of the developer kit, and sending their values over MQTT using the Paho client.

We made a live demo of a Raspberry Pi running Mihini connected to the developer kit at EclipseCon Europe at the end of October, and all the source code and schematics necessary for building your own will be released shortly. There is a page [3] on the Mihini wiki that you can refer to if you want to get information regarding the kit and when it will actually be available.

Targeting an Eclipse package for M2M developers for Kepler

We are aware that setting up Eclipse can sometimes be complex for newcomers, and we really want the developer experience to be as smooth as possible when using the M2M technologies developed at Eclipse. In order to simplify onboarding of developers, and provide them with all the tools they need for creating their applications, we are currently defining an all-in-one package that will, similarly to e.g. "Eclipse for C/C++ Developers", already contain all the plug-ins one might need for working with Eclipse M2M technologies. This IDE, which should be released with Kepler, will notably include Lua Development Tools with Mihini support and tools for debugging OMA-DM and MQTT communications.

m2m.eclipse.org: a new portal for learning more about the initiative

One of the major goals of the M2M Industry Working Group is really to make it easier for M2M developers to access all the tools, frameworks, and information they need for building their solutions. The recently launched m2m.eclipse.org website is meant as a one-stop-shop for learning more about the initiative: not only is it a window to the three Eclipse projects Koneki, Paho, and Mihini, but it is also where we plan on making available detailed tutorials, code samples, and of course the aforementioned M2M IDE.

Now you know where you should have a look if you want to learn more about Mihini and the initiative in general!

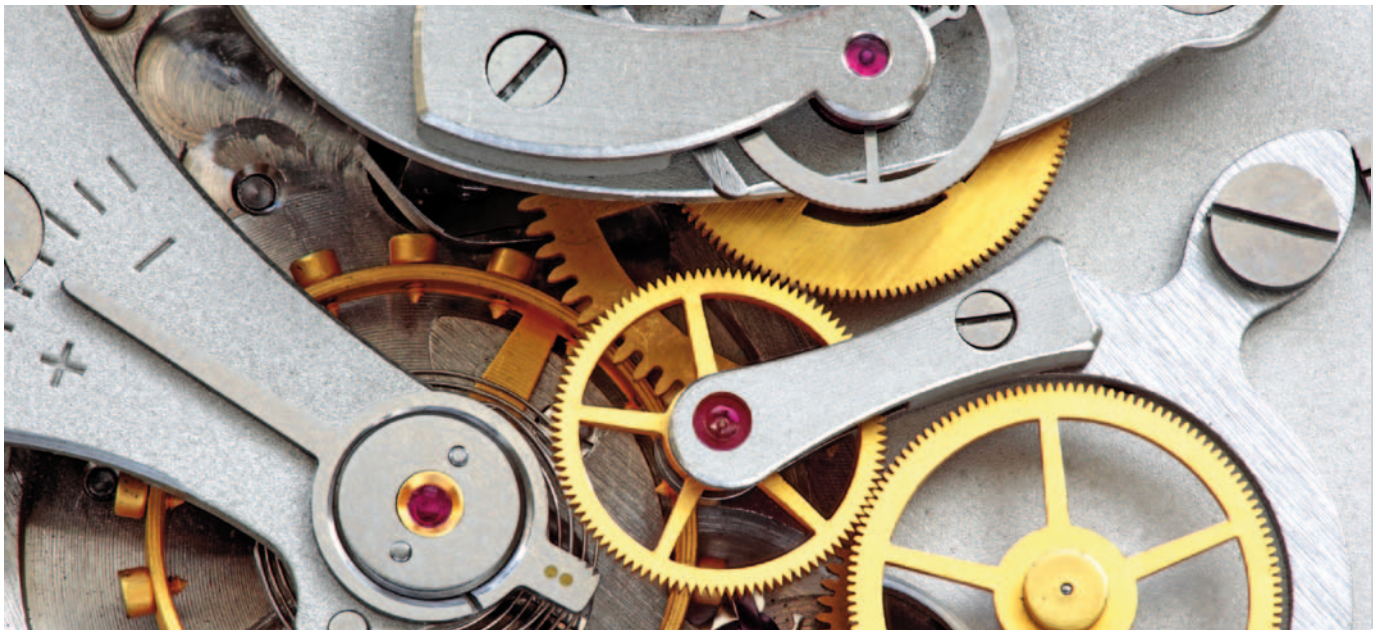
Benjamin is Open Source Evangelist at Sierra Wireless. He has a longtime passion for Eclipse and its ecosystem, and is a committer on several Eclipse projects (e4, PDE, ...) and contributor to numerous other open source projects. He leads the Koneki and Mihini projects and actively participates to the M2M Industry Working Group. In his day-to-day job, he supports the community and advocates the use of innovative technologies (Lua, modeling, ...) for the Internet of Things. When not wandering on the Koneki forum, he is building crazy communicating devices using Arduino kits! You can find him online on Twitter (@kartben) or on his blog: http://blog.benjamin-cabe.com.

References

[1] http://www.eclipse.org/koneki

[2] http://www.eclipse.org/mihini

[3] http://wiki.eclipse.org/Mihini/Developer_Kit



Carl Dea clocks some hours with Java's rich client technology, JavaFX

A Quick Glimpse of JavaFX's Canvas API

I recently attended the JavaOne 2012 conference hosted by Oracle this past October. My main goal was to focus on all things relating to client-side Java, even if the sessions didn't contain "JavaFX" in the title [1] [2]. To my surprise, I received much more than I bargained for. I attended sessions involving technologies such as Dolphin, Griffon, Java Embedded, JavaFX 8, etc [3], [4], [5].

by Carl Dea

I couldn't help but drool over the new JavaFX 8's 3D APIs. However, another one of my intentions was to look at the many new features that were added to the current release of JavaFX 2.2 [6]. In particular, one feature that comes to mind is the highly anticipated Canvas API. In this article, you will learn about some of the performance characteristics when using JavaFX's Canvas API [7], and later, I will share some code and teach you how to build an animated clock using only the Canvas API on JavaFX's scene graph as opposed to the vector based counterparts (JavaFX 2.1 and earlier).

Of the many JavaFX sessions at the JavaOne 2012 conference, one session, in particular, which stood out was (CON6784 – JavaFX Graphics Tips and Tricks) with Richard Bair. Mr. Bair, a JavaFX Architect, discussed JavaFX 2 and JavaFX 8's graphic rendering performance in comparison with all of the major browsers on popular desktop operating systems (Windows, Linux/Ubuntu, and Mac OS X) [8]. The benchmark (following the GUIMark 2) results were categorized by 'Vector' and 'Bitmap' graphics rendering strategies [9]. Familiar to most HTML5 Web developers, using the 'Vector' rendering strategy is simply synonymous to the SVG (XML representation of scalable vector graphics) [10]. In ad-

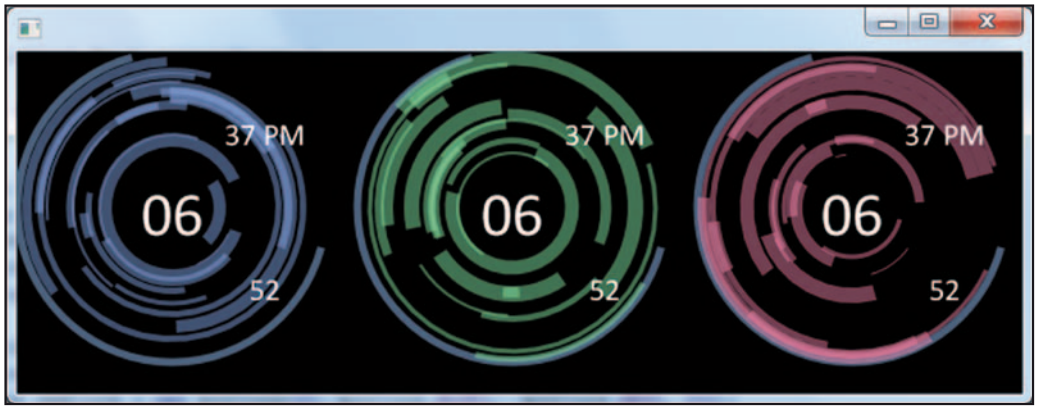


Figure 1: Tron Legacy style clocks with JavaFX

Figure 2: Caption

dition to this, the 'Bitmap' rendering strategy is equivalent to the use of the popular HTML5 Canvas API using the JavaScript language [11].

When involving the vector rendering strategy, the performance numbers showed that JavaFX had outperformed all of its competitors with the exception of the Safari browser on Mac OS X. Similarly, when involving the bitmap rendering strategy, the performance metrics indicate that the JavaFX Canvas API outperformed all the browsers by a significant amount. Mr. Bair did a great job in explaining when, why, and how to choose between the two rendering strategies. Mr. Bair also points out that, in some scenarios, JavaFX vector graphics can even outperform JavaFX canvas. He also suggested that, when using one technique over the other, the only way to really get a more accurate picture is to profile the application. Of course, much more was discussed in his talk. However, in this article I merely wish to point out the relevant bits of information relating to the Canvas API.

In the world of JavaFX, a scene graph is mainly based on the 'Vector' rendering strategy which calculates shapes and paths as scene graph nodes which are easily scaled. For those developers who recall prior to JavaFX 2.2, you will remember when there was not an ability to handle bitmap graphics (pixel pushing). You are probably wondering, "How do you render using the 'Bitmap' strategy onto the JavaFX vector based scene graph?" Since the release of JavaFX 2.2, the JavaFX team created the Canvas API. The Canvas API is actually a JavaFX node called the *Canvas* class. Since the *Canvas* class extends from *javafx.scene.Node* class, it is basically a scene graph node like any other JavaFX node, except you have the ability to draw directly and manipulate pixel data using graphics primitives (Canvas API) onto the surface (*GraphicsContext*).

Listing 1

```
Canvas canvas = new Canvas(300, 300);
final GraphicsContext gc = canvas.getGraphicsContext2D();
gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());

gc.setFill(Color.BLACK);
gc.setFont(Font.getDefault());
gc.fillText("hello   world!", 15, 50);

gc.setLineWidth(5);
gc.setStroke(Color.PURPLE);

gc.strokeOval(10, 60, 30, 30);
gc.strokeOval(60, 60, 30, 30);
gc.strokeRect(30, 100, 40, 40);

root.getChildren().add(canvas);
primaryStage.setScene(scene);
primaryStage.show();
```

Listing 2

```
final GraphicsContext gc = canvas.getGraphicsContext2D();
...
gc.setStroke(strokeColor);
gc.setLineWidth(strokeWidth);
gc.strokeArc(x,  // upper left x coord
      y,    // upper left y coord
      w,    // width
      h,    // height
      startAngle,
      arcExtent,
      ArcType.OPEN);
```

How do you draw on the Canvas node?

To draw on a Canvas, you must first instantiate a *Canvas* node with a width and height. After instantiating a *Canvas* class, you will need to obtain its *GraphicsContext* object [12]. To obtain the *GraphicsContext,* you simply invoke its *getGraphicsContext2D()* method. The *GraphicsContext* class has methods enable the developer to access many of the graphics drawing primitives. These drawing primitives provide ways to draw shapes, fill colors, set strokes, apply images, draw text, and much more. Please refer to the JavaDoc for details relating to the *GraphicsContext* class. Listing 1 demonstrates the basics of drawing onto the *GraphicsContext*. **Figure 2** shows.

When creating an instance of a *Canvas* class, you will have an opportunity to place it into the scene graph like any other JavaFX node. Since *Canvas* is a JavaFX *Node* class,

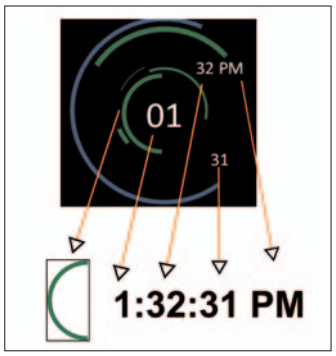


Figure 3: A visual design mockup comprised of the parts of a clock

the developer (user of the API) is free to treat it like any other node in the scene graph. This means that you have the ability to apply effects, transforms, and animations. One of the main advantages of the Canvas API is that you can draw as many things on the graphic context (bitmap surface) which is similar to the vector rendering strategy. However, instead of many nodes on the scene graph, all of the drawings on the canvas are encapsulated in one node, thus increasing performance. Remember, cutting down on the number of node objects on the scene graph helps to achieve better efficiencies such as graph walking. A downside to using the Canvas API is that the developer must draw things from scratch and handle efficiencies on their own (such as calculating dirty regions for optimization). This is why, in some cases, JavaFX's vector based strategy is able to excel in performance. These advancements in performance are due to its ability to detect dirty regions, offer caching hints and perform other tricks under the covers. Everything boils down to what you are really trying to achieve and the desired effect.

If you've dabbled with HTML5 development, you'll notice JavaFX's Canvas API was designed to look similar to HTML5's Canvas API (no coincidence). I was a bit surprised that the programming interface didn't follow the Java 2D API (of which I'm a big fan). However, I quickly got over it. Many of the popular HTML5/JavaScript examples on the net can be easily ported to the JavaFX Canvas API because of their similar naming conventions. (Note: In order to use the Canvas API, you must install JavaFX 2.2 or a later version.)

With all the excitement surrounding JavaFX's Canvas API, I decided to take it out for a spin. Since JavaFX's Canvas API

Listing 3

```
/** Driver class to run the demo which extends
 * from JavaFX's Application class. The demo will
 * make use of JavaFX's Canvas API. This class
 * creates three clocks which are animated using an
 * AnimationTimer class.
 *
 * Inspired by http://burnwell88.deviantart.com/
 * art/Clock-136761577
 * and http://rainmeter.net/cms/
 *
 * Created with IntelliJ IDEA.
 * User: cdea
 * Date: 10/15/12
 * Time: 12:20 AM
 *
 */
public class TronClockDemo extends Application {

    public static void main(String[] args) {
        Application.launch(args);
    }

    @Override
    public void start(final Stage primaryStage) {

        Group root = new Group();
        Scene scene = new Scene(root, 650, 220,
                                Color.rgb(0,0,0));

        // create a canvas node
        Canvas canvas = new Canvas();

        // bind the dimensions when the user resizes
        // the window.
        canvas.widthProperty().bind(primaryStage.
                                    widthProperty());
        canvas.heightProperty().bind(primaryStage.
                                    heightProperty());

        // obtain the GraphicsContext (drawing
        // surface)
        final GraphicsContext gc =
                    canvas.getGraphicsContext2D();

        // create three clocks
        final ArcClock blueClock = new ArcClock(20,
                                BLUE1, BLUE2, 200);
        final ArcClock greenClock = new ArcClock(20,
                               BLUE1, GREEN1, 200);
        final ArcClock redClock = new ArcClock(20,
                                BLUE1, RED1, 200);

        // create an animation (update & render loop)
        new AnimationTimer() {
            @Override
            public void handle(long now) {
                // update clocks
                blueClock.update(now);
                greenClock.update(now);
                redClock.update(now);

                // clear screen
                gc.clearRect(0, 0, primaryStage.
                                    getWidth(),
                        primaryStage.getHeight());

                // draw blue clock
                blueClock.draw(gc);
                // save the origin or the current state
                // of the Graphics Context.
                gc.save();

                // shift x coord position the width of a
                // clock plus 20 pixels
                gc.translate(blueClock.maxDiameter +
                                                20, 0);
                greenClock.draw(gc);

                // shift x coord position past the first
                // clock
                gc.translate(blueClock.maxDiameter +
                                                20, 0);
                redClock.draw(gc);

                // reset Graphics Context to last saved
                // point.
                // Translate x, y to (0,0)
                gc.restore();

            }
        }.start();

        // add the single node onto the scene graph
        root.getChildren().add(canvas);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

}
```

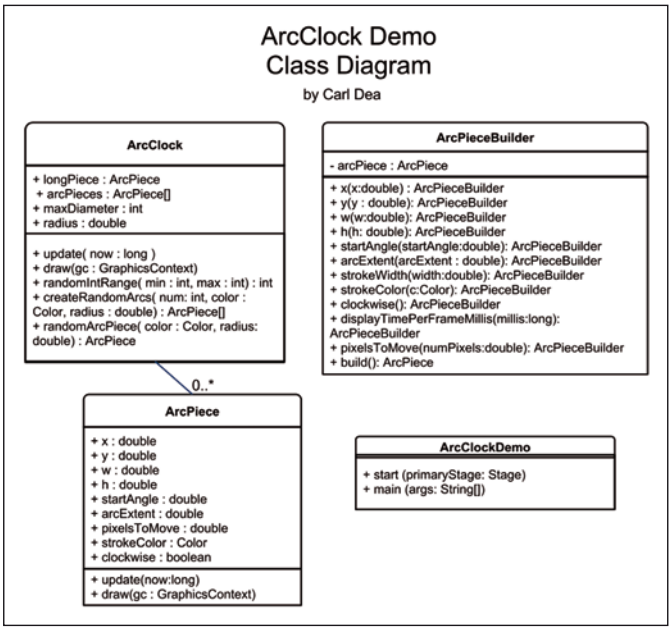


Figure 4: ArcClock Demo Class Diagram

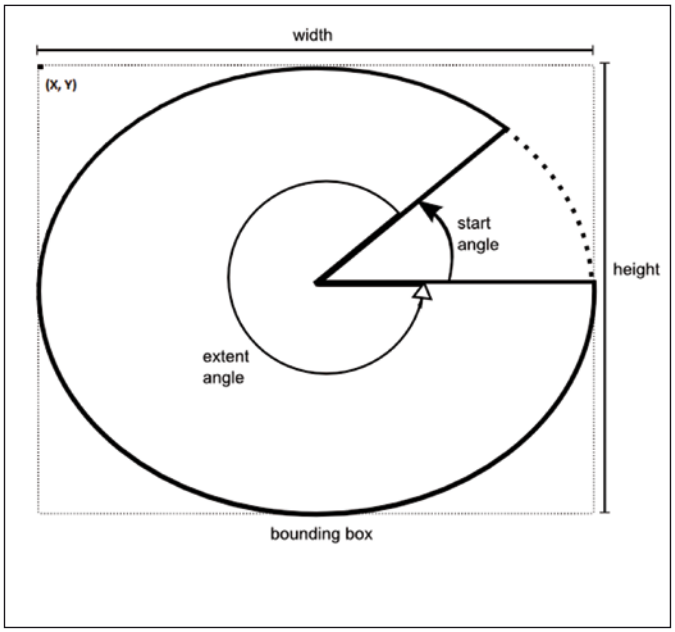


Figure 5: Arc Anatomy

has so many features, I will only give you a quick glimpse by presenting a futuristic stylized clock reminiscing of aspects of the movie Tron Legacy [13]. The clock was inspired by a user named Burnwell88 at DeviantArt [14] [15]. The clock was developed using the Rain Meter desktop theme maker [16]. In my concluding statements, I will list some of the many features related to the Canvas API that won't be discussed in this article.

Before we jump into the code, I want to show you a simple design of the clock and also provide a class diagram depicting the classes used in the demo. **Figure 3** depicts a mockup of the parts that comprise the arc clock (**Figure 3**).

Figure 3 describes the parts that comprise our demo 'arc clock' consisting of the following: An arc (*ArcPiece*), hours, minutes, seconds and meridian respectively (from left to right). Now that we visually understand what pieces make up an arc clock, let me show you a class diagram listing all of the classes we need to capture the requirements for the demo. **Figure 4** lists the four classes used in this clock demo.

The left side of the class diagram in **Figure 4** show two main classes used to draw a clock using arcs. The *ArcClock* class is a container-type class which holds anywhere from zero to many *ArcPiece* instances. An *ArcClock* also contains the clock's radius and diameter (bounding box width of the clock). The individual arc objects (*ArcPiece)* will contain the arc attributes which will be later rendered onto the canvas with each update and render cycle. In **Figure 3**, the *ArcPieceBuilder* class (upper right) is a convenience class that follows an object oriented builder pattern [17]. Using this class provides an easy way to construct an *ArcPiece* instance in declarative syntax style programming. This pattern enables the developer to reduce boilerplate code and allows ad hoc parameters to be specified (method chaining). Table 1 displays all the attributes which describes an *ArcPiece* class.

For brevity's sake, I felt it was better to expose attributes as *public* and not to implement getter and setter methods which are similar to the Java bean specification (conven-

Attribute	Data Type	Example	Description
x	double	0	Upper left x coordinate of the bounding box
y	double	0	Upper left y coordinate of the bounding box
w	double	200	Width of the bounding box surrounding clock
h	double	200	Height of the bounding box surrounding clock
startAngle	double	45	Starting angle of the arc in degrees measure
arcExtent	double	240	Extent angle in degrees measure
strokeWidth	double	5	The pixel thickness of the stroke of the arc line
pixelsToMove	double	1	Number of pixels to animate arc appearing to rotate
strokeColor	Color	Color.RED	The color of the stroke
clockwise	boolean	true	Animate arc to move clockwise otherwise counter clockwise

Table 1: The attributes contained in the "ArcPiece" class

tion). Also, I chose to use simple primitive data types instead of JavaFX Properties APIs. Using primitives can help lower overhead (the need to bind against many values). At some point, you may want to run the demo with an embedded version of Java on a more constrained device such as Raspberry Pi, BeagleBoard, etc [18] [19].

Now that you know the attributes to draw and arc, let's see what it would look like. **Figure 5** depicts an arc drawn using the attributes x, y, w, h, startAngle, arcExtent, strokeColor, and *ArcType*. For our demo, I have hard-coded all the arcs to be *ArcType.OPEN*. **Figure 5** shows the information of an arc as it is to be drawn on the canvas.

Listing 2 shows the code snippet used to draw an arc onto the *GraphicsContext*.

The main driver or JavaFX Application which runs our demo is housed in the *TronClockDemo* class (Listing 3). This main driver class contains an *AnimationTimer* instance which will periodically update the arc clock's attributes and draw the parts onto the *GraphicsContext* surface [20]. In Listing 3, the main driver class, TronClockDemo.java, launches the JavaFX application.

Initially, the application's main method will launch the JavaFX application thread via the *Application.launch()* method. Once the application thread is ready (post *init()* method), the *start()* method gets invoked. To see other lifecycle methods, go to the JavaDocs on *Application class* [21]. You'll notice in Listing 3 that all the action occurs in the *start()* method. This is where an *AnimationTimer* instance is created using an anonymous inner class and started using the *AnimationTimer.start()* method. As each cycle occurs, the *handle()* method is invoked by the JavaFX Application thread. When invoking the *handle()* method, you'll notice the inbound parameter '*now*' of type *long* is the current time in nanoseconds for one frame.

The body of the handle() method will take the following steps:

1. Update clock models (blue clock, green clock, and red clock)
2. Clear screen
3. Draw blue clock
4. Save GraphicsContext state
5. Translate X coordinates to the right of the blue clock
6. Draw the green clock
7. Translate X coordinates to the right of the green clock
8. Draw the red clock
9. Restore or return to the last save GraphicsContext state

Attribute	Data Type	Example	Description
longPiece	ArcPiece		A single open arc drawn on the outer rim of the clock
arcPieces	ArcPiece		Many randomly drawn open arcs in the clock
maxDiameter	int	200	Width of the clock
radius	double	100	Half of the width of the clock

Table 2: The attributes contained in the "ArcClock" class

Listing 4: ArcPiece.java

```
/** ArcPiece is an object representing the state
 * (model) of an arc shape that will be drawn on
 * the Graphics Context. During
 * an animation loop values in the model will often
 * be updated. The update() method has the ability
 * to calculate elapsed time to allow the arc to later
 * be animated based on
 * frames per second. The draw() method simply
 * renders the arc
 * shape onto the Graphics Context (surface).
 *
 * Created with IntelliJ IDEA.
 * User: cdea
 */
public class ArcPiece {
    public double x;
    public double y;
    public double w;
    public double h;
    public double startAngle;
    public double arcExtent;
    public double strokeWidth = 2;
    public double pixelsToMove = 2;
    public Color strokeColor;
    public boolean clockwise=false;

    long startTime = 0;
    public long displayTimePerFrameMillis = 60;
    private long displayTimePerFrameNano = 60  *
                                        1000000;

    public void update(long now) {
        if (startTime == 0){
            startTime = now;
            displayTimePerFrameNano =
                    displayTimePerFrameMillis * 1000000;
        }

        long elapsed = now - startTime;
        if (elapsed > displayTimePerFrameNano) {
            if (!clockwise){
                startAngle = startAngle + pixelsToMove;
                if (startAngle > 360){
                    startAngle = 0;
                }
            } else {
                startAngle = startAngle - pixelsToMove;
                if (startAngle < -360){
                    startAngle = 0;
                }
            }
            startTime = 0;
        }
    }

    public void draw(GraphicsContext gc) {
        gc.setStroke(strokeColor);
        gc.setLineWidth(strokeWidth);
        gc.strokeArc(x,
                y,
                w,
                h,
                startAngle,
                arcExtent,
                ArcType.OPEN);
    }
}
```

Listing 5: ArcClock.java

```
/** This class is a container class that maintains
 * all of the parts that comprised of a clock. It is
 * also responsible for updating and drawing itself
 * at each animation frame cycle.
 * The ArcClock contains zero to many ArcPiece objects
 * to be drawn. The clock will also display the time
 * of day.
 *
 * User: cdea
 */
public class ArcClock {
    public static Color BLUE1 = Color.rgb(126, 166, 212, 0.6);
    public static Color BLUE2 = Color.rgb(126, 166, 222, 0.5);
    public static Color BLUE3 = Color.rgb(130, 166, 230, 0.5);
    public static Color GREEN1 = Color.rgb(130, 230, 166, 0.5);
    public static Color RED1 = Color.rgb(230, 130, 166, 0.5);

    public ArcPiece longPiece;
    public ArcPiece[] arcPieces;
    public int maxDiameter;
    public double radius;

    private static final DateFormat DATE_FORMAT = new
                                        SimpleDateFormat("yyyyMMddhhmmssa");

    public ArcClock(int numArcs, Color longPieceColor, Color manyPieceColor, int
                                                                maxDiameter) {
        this.maxDiameter = maxDiameter;
        radius = maxDiameter / 2;
        longPiece = ArcPieceBuilder.create()
                .strokeColor(longPieceColor)
                .strokeWidth(5)
                .x(0)
                .y(0)
                .w(maxDiameter)
                .h(maxDiameter)
                .startAngle(45)
                .arcExtent(240)
                .displayTimePerFrameMillis(1000)
                .pixelsToMove(1)
                .build();

        arcPieces = createRandomArcs(numArcs, manyPieceColor, maxDiameter / 2);
    }

    public void update(long now){
        longPiece.update(now);
        for (ArcPiece ap:arcPieces) {
            ap.update(now);
        }
    }

    public void draw(GraphicsContext gc) {
        longPiece.draw(gc);
        for (ArcPiece ap:arcPieces) {
            ap.draw(gc);
        }
        // draw hour
        gc.setFont(Font.font("Calibri", 40));
        gc.setFill(Color.WHITE);
        gc.setTextAlign(TextAlignment.CENTER);

        String dateTimeStr = DATE_FORMAT.format(new Date());
        //yyyyMMddhhmmssa
        gc.fillText(dateTimeStr.substring(8, 10) , radius, radius + 18 );
        gc.setFont(Font.font("Calibri", 20));
        gc.fillText(dateTimeStr.substring(10, 12) + " " + dateTimeStr.substring(14) ,
                                                    maxDiameter - 40, radius - 40 );
        gc.fillText(dateTimeStr.substring(12, 14) , maxDiameter - 40,
                                                                maxDiameter - 40 );

    }

    public static int randomIntRange(int min, int max) {
        Random rand = new Random();
        int range = max - min + 1;
        return rand.nextInt(range) + min;
    }

    public static ArcPiece[] createRandomArcs(int num, Color color, double radius)
{
        final ArcPiece[] manyPieces = new ArcPiece[num];
        for (int i=0; i<num; i++) {
            manyPieces[i] = randomArcPiece(color, radius);
        }
        return manyPieces;
    }
    public static ArcPiece randomArcPiece(Color color, double radius) {

        int width =  randomIntRange(60, (int) radius * 2);
        int randomStrokeWidth = randomIntRange(1,10);
        int randomStartAngle = randomIntRange(1, 270);
        int randomExtentAngle = randomIntRange(10, 360-randomStartAngle);
        long randomMillis = randomIntRange(0, 33);
        Color someColor = color;
        if (color == null) {
            someColor =  BLUE1;
        }
        final ArcPiece arcPiece = ArcPieceBuilder.create()
                .strokeColor(someColor)
                .strokeWidth(randomStrokeWidth)
                .x(radius - (width/2))
                .y(radius - (width/2))
                .w(width)
                .h(width)
                .startAngle(randomStartAngle)
                .arcExtent(randomExtentAngle)
                .displayTimePerFrameMillis(randomMillis)
                .pixelsToMove(2)
                .build();
        arcPiece.clockwise = new Random().nextBoolean();

        return arcPiece;
    }
}
```

Listing 4 shows the source code for a simple Java class which is responsible for containing an arc's model.

Listing 4 also contains two methods *update()* and *draw()* which will assist the animation cycle to update the values and render the arc onto the drawing surface (*GraphicsContext*).

Listing 5 displays the source code for the *ArcClock.java* file which is the container class representing the model of a clock.

Listing 5 begins with the predefined colors that will be used in our three demo clocks. The instance variables are the following: *longPiece*, *arcPieces*, *maxDiameter*, and *radius*. Table 2 describes the attributes of an instance of an *ArcClock* class.

To create an instance of an *ArcClock*, the constructor will call many static methods to build random open arcs shapes with varied start angles and extent angles. Another thing to point out is the *update()* and *draw()* methods which are used to assist in the animation cycle (*AnimationTimer*). At each animation cycle, the clock will update the model and render the arcs onto the graphics context. In addition to rendering all the arc pieces for the arc clock, the *ArcClock* class will also display (draw) the time using the *GraphicsContext's fillText()* method.

Well, there you have it, a quick glimpse of the JavaFX Canvas API. Although we've only touched the surface of the capabilities of the Canvas API, I have listed some of the other interesting features of JavaFX 2.2 and the Canvas API.

- Snapshot – Any Scene graph node can capture a node's as a bitmap image.
- PixelWriter – Ability to modify pixel information on a bitmap image.
- PixelReader – Ability to read pixel information from a bitmap image.
- Blending – Blending modes when drawing on the canvas (graphics context).
- Layers – Ability to have many graphics contexts to provide layers (z order).

Happy coding! I trust you will dig deeper into these interesting features.

Carl P. Dea is currently a Sr. Software Engineer, co-author of Java 7 Recipes, author of JavaFX 2 Introduction by Example, and technical reviewer of Pro JavaFX 2 from Apress publishing. Carl has been developing software for over 15 years with many clients from fortune 500 companies to nonprofit organizations. He has written software ranging from mission critical applications to e-commerce-based Web applications. Carl has been using Java since the very beginning and he also is a huge JavaFX enthusiast dating back when it used to be called F3. His current software development interests are: UI/UX, game programming, data visualization, microcontrollers, smart phones and tablet computers. When he's not working he and his wife love to watch their daughters perform at gymnastics meets. Carl lives on the East Coast in Pasadena, Maryland USA. He blogs at http://carlfx.wordpress.com. Carl is alsoconnected to LinkedIn at http://www.linkedin.com/in/carldea and is a Twitter user @carldea.

References

[1] JavaOne: http://www.oracle.com/javaone

[2] Oracle corp.: http://www.oracle.com

[3] Dolphin: http://www.canoo.com/blog/2012/10/01/canoo-announces-open-sourcing-of-dolphin

[4] Griffon: http://griffon.codehaus.org

[5] Java Embedded: http://www.oracle.com/javaone/embedded

[6] JavaFX 2.2: http://docs.oracle.com/javafx/2/release_notes_2-2/jfxpub-release_notes_2-2.htm

[7] JavaFX Canvas API: http://docs.oracle.com/javafx/2/api/javafx/scene/canvas/Canvas.html

[8] CON6784, JavaFX Graphics Tips and Tricks: https://oracleus.activeevents.com/connect/sessionDetail.ww?SESSION_ID=6784

[9] GUIMark 2: http://www.craftymind.com/guimark2/

[10] SVG: http://www.w3.org/Graphics/SVG/

[11] HTML5 Canvas API: http://www.w3schools.com/html/html5_canvas.asp

[12] GraphicsContext API: http://docs.oracle.com/javafx/2/api/javafx/scene/canvas/GraphicsContext.html

[13] Tron Legacy: http://www.imdb.com/title/tt1104001/

[14] Burnwell88 at deviantart.com: http://burnwell88.deviantart.com/gallery/#/d29f9wp

[15] DeviantArt.com: http://deviantart.com

[16] Rain Meter: http://rainmeter.net/

[17] Builder Pattern: http://blog.netopyr.com/2012/01/24/advantages-of-javafx-builders/

[18] Raspberry Pi: http://www.raspberrypi.org/

[19] BeagleBoard: http://beagleboard.org/

[20] AnimationTimer: http://docs.oracle.com/javafx/2/api/javafx/animation/AnimationTimer.html

[21] JavaFX Application API: http://docs.oracle.com/javafx/2/api/javafx/application/Application.html

[22] JavaFX 2.2 is here, and JavaFX 8.0 is on its way!: http://fxexperience.com/2012/08/javafx-2-2-is-here-and-javafx-8-0-is-on-its-way/

[23] Working with Canvas by Scott Hommel: http://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm

[24] Dead bitmaps could be beautiful by Gerrit Grunwald: http://harmoniccode.blogspot.de/2012/05/death-bitmaps-could-be-beautiful-part-i.html

[25] Take care of the JavaFX scene graph...: http://www.canoo.com/blog/2012/09/21/take-care-of-the-javafx-scene-graph

[26] Java 7 recipes, Chapter 12 Figure 4 depicting an Arc: http://www.apress.com/9781430240563

[27] JavaFX 2 Introduction by Example: http://www.apress.com/9781430242574

[28] Pro JavaFX 2: http://www.apress.com/9781430268727

The FREE technical whitepapers resource

Java

Big Data

Mobile

.NET

Cloud

Web

For more information visit: **www.whitepapers360.com**

Using anonymous actors to capture context while processing asynchronous tasks

Asynchronous Programming With Akka Actors

One of the most difficult tasks in asynchronous programming is trying to capture context so that the state of the world at the time the task was started can be accurately represented at the time the task finishes.

by Jamie Allen

However, creating anonymous instances of Akka actors is a very simple and lightweight solution for capturing the context at the time the message was handled to be utilized when the tasks are successfully completed [1].

The Problem

A great example is an actor which is sequentially handling messages in its mailbox but performing the tasks based on those message off-thread with Futures. This is a great way to design your actors in that they will not block waiting for responses, allowing them to handle more messages concurrently and increase your application's performance. However, the state of the actor will likely change with every message.

Let's take an example of an actor which will act as a proxy to get a customer account information for a financial services firm from multiple data sources. Further, let's assume that each of the subsystem proxies for savings, checking and money market account balances will optionally return a list of accounts and their balances of that kind for this customer. Let's write some basic Akka actor code to perform this task:

This code is fairly concise. The AccountBalanceRetriever actor receives a message to get account balances for a customer, and then it fires off three futures in parallel. The first will get the customer's savings account balance, the second will get the checking account balance and the third will get a money market balance. Doing these tasks in parallel allows us to avoid the expensive cost of performing the retrievals sequentially. Also, note that while the futures will return Options of some account balances by account ID, if they return None, they will not short-circuit the for comprehension – if None is returned from futSavings, it will still continue the for comprehension.

However, there are a couple of things about it that are not ideal. First of all, it is using futures to ask other actors for responses and creating a new PromiseActorRef for every message sent, which is a waste of resources. It would be better to have our AccountBalanceRetriever actor send messages out in a "fire and forget" fashion and collect results asynchronously into *one* actor..

Furthermore, there is a glaring race condition in this code – can you see it? We're referencing the "sender" in our map operation on the result from futBalances, which may not be the same ActorRef when the future completes, because the AccountBalanceRetriever ActorRef may now be handling another message from a different sender at that point!

Avoiding Ask

Let's focus on eliminating the need to ask for responses in our actor first. We can send the messages with the "!" and collect responses into a List of an optional List of balances by account number. But how would we go about doing that?

This is better, but still leaves a lot to be desired. First of all, we've created our collection of balances we've received back at the instance level, which means we can't differentiate the aggregation of responses to a single request to get account balances. Worse, we can't time out a request back to our original requestor. Finally, while we've captured the original sender as an instance variable that may or may not have a value (since there is no originalSender when the AccountBalanceRetriever starts up), we have no way of being sure that the originalSender is who we want it to be when we want to send data back!

Listing 1

```
import scala.concurrent.ExecutionContext
import scala.concurrent.duration._
import akka.actor._
import akka.pattern.ask
import akka.util.Timeout

case class GetCustomerAccountBalances(id: Long)
case class AccountBalances(
 val checking: Option[List[(Long, BigDecimal)]],
 val savings: Option[List[(Long, BigDecimal)]],
 val moneyMarket: Option[List[(Long,
                                    BigDecimal)]])
case class CheckingAccountBalances(
 val balances: Option[List[(Long, BigDecimal)]])
case class SavingsAccountBalances(
 val balances: Option[List[(Long, BigDecimal)]])
case class MoneyMarketAccountBalances(
 val balances: Option[List[(Long, BigDecimal)]])

class SavingsAccountProxy extends Actor {
 def receive = {
  case GetCustomerAccountBalances(id: Long) =>
   sender ! SavingsAccountBalances(Some(List((1,
                          150000), (2, 29000))))
 }
}
class CheckingAccountProxy extends Actor {
 def receive = {
  case GetCustomerAccountBalances(id: Long) =>
   sender ! CheckingAccountBalances(Some(List((3,
                                    15000))))
 }
}
class MoneyMarketAccountsProxy extends Actor {
 def receive = {
  case GetCustomerAccountBalances(id: Long) =>
   sender ! MoneyMarketAccountBalances(None)
 }
}

class AccountBalanceRetriever(savingsAccounts:
            ActorRef, checkingAccounts: ActorRef,
    moneyMarketAccounts: ActorRef) extends Actor {
 implicit val timeout: Timeout = 100 milliseconds
 implicit val ec: ExecutionContext = context.
dispatcher
 def receive = {
  case GetCustomerAccountBalances(id) =>
   val futSavings = savingsAccounts ?
                    GetCustomerAccountBalances(id)
   val futChecking = checkingAccounts ?
                    GetCustomerAccountBalances(id)
   val futMM = moneyMarketAccounts ?
                    GetCustomerAccountBalances(id)
   val futBalances = for {
    savings <- futSavings.mapTo[Option[List[(Long,
                                    BigDecimal)]]]
    checking <- futChecking.mapTo[Option[List[(Long,
                                    BigDecimal)]]]
    mm <- futMM.mapTo[Option[List[(Long,
                                    BigDecimal)]]]
   } yield AccountBalances(savings, checking, mm)
   futBalances map (sender ! _)
 }
}
```

Listing 2

```
import scala.concurrent.ExecutionContext
import scala.concurrent.duration._
import akka.actor._

case class GetCustomerAccountBalances(id: Long)
case class AccountBalances(
  val checking: Option[List[(Long, BigDecimal)]],
  val savings: Option[List[(Long, BigDecimal)]],
  val moneyMarket: Option[List[(Long,
                                    BigDecimal)]])
case class CheckingAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])
case class SavingsAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])
case class MoneyMarketAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])

class SavingsAccountProxy extends Actor {
  def receive = {
    case GetCustomerAccountBalances(id: Long) =>
     sender ! SavingsAccountBalances(Some(List((1,
                          150000), (2, 29000))))
  }
}
class CheckingAccountProxy extends Actor {
 def receive = {
  case GetCustomerAccountBalances(id: Long) =>
   sender ! CheckingAccountBalances(Some(List((3,
                                    15000))))
  }
}
class MoneyMarketAccountsProxy extends Actor {
 def receive = {
  case GetCustomerAccountBalances(id: Long) =>
   sender ! MoneyMarketAccountBalances(None)
 }
}

class AccountBalanceRetriever(savingsAccounts:
            ActorRef, checkingAccounts: ActorRef,
    moneyMarketAccounts: ActorRef) extends Actor {
  val checkingBalances, savingsBalances,
mmBalances: Option[List[(Long,
                                    BigDecimal)]] = None
  var originalSender: Option[ActorRef] = None
  def receive = {
   case GetCustomerAccountBalances(id) =>
    originalSender = Some(sender)
    savingsAccounts !
                    GetCustomerAccountBalances(id)
    checkingAccounts !
                    GetCustomerAccountBalances(id)
    moneyMarketAccounts !
                    GetCustomerAccountBalances(id)
   case AccountBalances(cBalances, sBalances,
                                    mmBalances) =>
    (checkingBalances, savingsBalances,
                                mmBalances) match {
     case (Some(c), Some(s), Some(m)) =>
     originalSender.get !
                AccountBalances(checkingBalances,
                                    savingsBalances,
      mmBalances)
     case _ =>
    }
  }
}
```

Capturing Context

The problem is that we're attempting to take the result of the off-thread operations of retrieving data from multiple sources and return it to whomever sent us the message in the first place. However, the actor will likely have moved on to handling additional messages in its mailbox by the time these futures complete, and the state represented in the AccountBalanceRetriever actor for "sender" at that time could be a completely different actor instance. So how do we get around this?

The trick is to create an anonymous inner actor for each GetCustomerInfo message that is being handled. In doing so, you can capture the state you need to have available when the futures are fulfilled. Let's see how.

This is much better. We've captured the state of each receive and only send it back to the originalSender when all three have values. But there are still two issues here. First, we haven't defined how we can time out on the original request for all of the balances back to whomever requested them. Secondly, our originalSender is STILL getting a wrong value – the "sender" from which it is assigned is actually the sender value of the anonymous inner actor, NOT the one that sent the original GetCustomerAccountBalances message!

Using a Promise

We can use a Promise to handle our need to timeout the original request, by allowing another task to compete for the right to complete the operation with a timeout. Promise in Scala

Listing 3

```
import scala.concurrent.ExecutionContext
 import scala.concurrent.duration._
 import akka.actor._

 case class GetCustomerAccountBalances(id: Long)
 case class AccountBalances(
  val checking: Option[List[(Long, BigDecimal)]],
  val savings: Option[List[(Long, BigDecimal)]],
  val moneyMarket: Option[List[(Long, BigDecimal)]])
 case class CheckingAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])
 case class SavingsAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])
 case class MoneyMarketAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])

 class SavingsAccountProxy extends Actor {
  def receive = {
   case GetCustomerAccountBalances(id: Long) =>
    sender ! SavingsAccountBalances(Some(List((1, 150000), (2, 29000))))
  }
 }
 class CheckingAccountProxy extends Actor {
  def receive = {
   case GetCustomerAccountBalances(id: Long) =>
    sender ! CheckingAccountBalances(Some(List((3, 15000))))
  }
 }
 class MoneyMarketAccountsProxy extends Actor {
  def receive = {
   case GetCustomerAccountBalances(id: Long) =>
    sender ! MoneyMarketAccountBalances(None)
  }
 }

 class AccountBalanceRetriever(savingsAccounts: ActorRef, checkingAccounts:
                               ActorRef, moneyMarketAccounts: ActorRef) extends Actor {
  val checkingBalances, savingsBalances, mmBalances: Option[List[(Long,
                                                      BigDecimal)]] = None
  var originalSender: Option[ActorRef] = None
  def receive = {
   case GetCustomerAccountBalances(id) => {
    context.actorOf(Props(new Actor() {
     var checkingBalances, savingsBalances, mmBalances: Option[List[(Long,
                                                      BigDecimal)]] = None
     val originalSender = sender
     def receive = {
      case CheckingAccountBalances(balances) =>
       checkingBalances = balances
       isDone
      case SavingsAccountBalances(balances) =>
       savingsBalances = balances
       isDone
      case MoneyMarketAccountBalances(balances) =>
       mmBalances = balances
       isDone
     }

     def isDone =
      (checkingBalances, savingsBalances, mmBalances) match {
       case (Some(c), Some(s), Some(m)) =>
        originalSender ! AccountBalances(checkingBalances, savingsBalances,
                                                      mmBalances)
        context.system.stop(self)
       case _ =>
      }

     savingsAccounts ! GetCustomerAccountBalances(id)
     checkingAccounts ! GetCustomerAccountBalances(id)
     moneyMarketAccounts ! GetCustomerAccountBalances(id)
    }))
   }
  }
 }
```

holds the expected return value as an Either, typed as Throwable if something goes wrong and as a generic type of whatever successful response you expected. In this case, we want an AccountBalances instance, so our Promise will be typed as Promise[AccountBalance].

Now we can collect our results and check to see if we successfully completed the promise. If so, we can return the appropriate value or the TimeoutException instance. Finally, we must remember to stop our anonymous inner actor so that we do not leak memory for every GetCustomerAccountBalances message we receive!

Conclusion

Asynchronous programming is simply not easy, even with powerful tools at our disposal. We always must think about the state we need and the context from which we get it. I hope this article has shown you some nice ideas about how to use Actors, Futures and Promises to perform complex tasks, as well as patterns for using them that will help you in your everyday coding. All source code can be found in my GitHub repository [2].

Jamie Allen has over 18 years of experience delivering enterprise solutions across myriad industries, platforms, environments and languages. He has been developing enterprise applications with Scala since 2009, primarily using actors for fault tolerance and managing concurrency at scale. Jamie currently works for Typesafe, where he helps users develop actor-based systems using the Akka framework and Scala.

References

[1] http://akka.io/

[2] https://github.com/jamie-allen/effective_akka

Listing 4

```
import java.util.concurrent.TimeoutException
import scala.concurrent.{ExecutionContext,
                                    Promise}
import scala.concurrent.duration._
import akka.actor._
import scala.math.BigDecimal.int2bigDecimal

case class GetCustomerAccountBalances(id: Long)
case class AccountBalances(
  val checking: Option[List[(Long, BigDecimal)]],
  val savings: Option[List[(Long, BigDecimal)]],
  val moneyMarket: Option[List[(Long,
                                  BigDecimal)]])
case class CheckingAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])
case class SavingsAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])
case class MoneyMarketAccountBalances(
  val balances: Option[List[(Long, BigDecimal)]])

class SavingsAccountProxy extends Actor {
  def receive = {
    case GetCustomerAccountBalances(id: Long) =>
      sender ! SavingsAccountBalances(Some(List((
                        1, 150000), (2, 29000))))
  }
}
class CheckingAccountProxy extends Actor {
  def receive = {
    case GetCustomerAccountBalances(id: Long) =>
      sender ! CheckingAccountBalances(
                       Some(List((3, 15000))))
  }
}
class MoneyMarketAccountsProxy extends Actor {
  def receive = {
    case GetCustomerAccountBalances(id: Long) =>
      sender ! MoneyMarketAccountBalances(None)
  }
}

class AccountBalanceRetriever(savingsAccounts:
            ActorRef, checkingAccounts: ActorRef,
  moneyMarketAccounts: ActorRef) extends Actor {
  def receive = {
    case GetCustomerAccountBalances(id) => {
      val originalSender = sender
      implicit val ec: ExecutionContext =
                                  context.dispatcher

      context.actorOf(Props(new Actor() {
        val promisedResult =
                      Promise[AccountBalances]()
        var checkingBalances, savingsBalances,
                  mmBalances: Option[List[(Long,
                              BigDecimal)]] = None
        def receive = {
          case CheckingAccountBalances(balances) =>
            checkingBalances = balances
            collectBalances
          case SavingsAccountBalances(balances) =>
            savingsBalances = balances
            collectBalances
          case MoneyMarketAccountBalances(
                                        balances) =>
            mmBalances = balances
            collectBalances
        }

        def collectBalances = (checkingBalances,
            savingsBalances, mmBalances) match {
          case (Some(c), Some(s), Some(m)) =>
            if (promisedResult.trySuccess(Account
          Balances(checkingBalances, savingsBalances,
                                    mmBalances)))
              sendResults
          case _ =>
        }

        def sendResults = {
          originalSender ! ((promisedResult.future.
                    map(x => x)) recover { case t:
                        TimeoutException => t })
          context.system.stop(self)
        }

        savingsAccounts !
                  GetCustomerAccountBalances(id)
        checkingAccounts !
                  GetCustomerAccountBalances(id)
        moneyMarketAccounts !
                  GetCustomerAccountBalances(id)
        context.system.scheduler.scheduleOnce(250
                                    milliseconds) {
          if (promisedResult.tryFailure(new
                                TimeoutException))
            sendResults
        }
      }))
    }
  }
}
```

Deciphering the term

Software Craftsmanship Revisited

Software Craftsmanship is a movement that gained a lot of momentum and attention in recent years. Its claim is that if you stick to its values, you will become a better software developer. Yet, the usual interpretation of the term has limitations. This article will take a closer look at the limitations and how to overcome them.

by Uwe Friedrichsen

The Manifesto for Software Craftsmanship: The term *Software Craftsmanship* gained a lot of attention in the recent years. The origin of this movement can be found in the *Manifesto for Software Craftsmanship* [1]. This manifesto was designed and written down by people who became convinced that the agile values and principles in themselves are not sufficient. They were convinced that it is not sufficient for good software development to be just agile, but that you – casually speaking – also have to be proficient in your trade.

Accordingly the manifesto is expressed as a continuation of the *Manifesto for Agile Software Development* [2]. As the agile manifesto it uses a left-side-right-side presentation. The form used is:

"Not only <value of agile manifesto>, but also <extended value of craftsmanship manifesto>"

This way it was possible to pick up and refine the four values of the agile manifesto. The four values the craftsmanship manifesto calls for are:

- well-crafted software
- steadily adding value
- community of professionals
- productive partnerships

It may look like a simple refinement of the agile manifesto (and indeed it came into existence this way), but actually it is much more than that. Even if the manifesto was designed as a result of dissatisfaction with the way the agile values were implemented in practice, Software Craftsmanship goes far beyond Agile and therefore appeals to many people who don't find Agile appealing.

The Usual Interpretation...

For me, the manifesto and the values associated with it are very valuable and very important because in my opinion they push the right ideas – values we badly need to be successful in an IT world where complexity and pressure to deliver always rise.

Great, so that's it! We are done, aren't we? Let's just stick to the values of the manifesto and we are set for our upcoming IT life. So, let's go to a local craftsmanship community nearby and learn from them. What are the typical things you will learn there?

Learn lifelong. Yes, this is important for sure, especially in a domain moving as fast as the IT domain. But actually, that is not new wisdom. All of us have heard that before but admittedly we cannot be reminded of it often enough. Fine, what else? Maybe something more concrete.

Do *Coding Katas* or *Code Retreats* or *Code Camps*. Do *TDD* (Test-Driven Development) and Clean Code. And things alike. Ah, okay? And what are those things good for? They help you to become a better software developer according to the values of the manifesto. Okay, that looks like fun. Let's get started with it. Let's do some katas, let's learn about *Test First, Baby Steps* and so on [3].

So you get started and it feels good. And you get new insights about writing good code. Then, why am I talking about limitations in the abstract in the article?

...and its limitations

Before I start to discuss the limitations of the usual interpretation, I would like to emphasize that in my opinion there is nothing wrong with the topics mentioned. On the contrary, there is a lot of value associated with coding katas, code retreats, TDD and all that. You should definitely become familiar with those concepts if you are not yet.

But if the whole discussion only revolves around enhancing coding skills, then there is something wrong. Why? The answer is twofold (see **Figure 1**). Let me start with the first dimension.

The eternal apprentice

Coding Katas and Code Retreats are usually smaller exercises to train a certain basic coding skill. This kind of exercise definitely makes sense for a junior software developer, but if I am still stuck with coding katas after 5 or 10 years of software development experience, I probably did not grow much as a software developer.

In martial arts, where the term *Kata* comes from, katas are strict series of movements that students use to train specific movement patterns. Depending on the martial arts style there are more or less katas, comprising of different levels of difficulty, but all of the styles have in common that masters do not improve by katas anymore.

A martial arts master must be able to select his or her movements according to the given situation, to any arbitrary attack pattern without following a strict series of predefined movements. You cannot learn that by only doing katas all the time. Katas for sure help to do moves right but they do not help to do the right (appropriate) moves to counter an arbitrary attack pattern. A master who only can reproduce predefined movement patterns is not a master but still a student, no matter how perfectly he or she executes the moves. And in most martial arts styles masters do not use katas to improve their skills but they have found different ways for themselves how to do it.

Again, there is nothing wrong with coding katas and code retreats, but if that is all we do to improve, then – using the analogy from trading – we will stay apprentices forever. We do not become a journeyman – not to mention a craftsman master at all.

Violating the values

If that were the only limitation we could just shrug and say, that limiting oneself this way is not very worthwhile but also not serious, even not unusual: Staying behind one's own potential is waste for sure but we see this all over the place. So what? Bad luck, we could say, if there were not the second dimension: The limitation described before violates the values of the manifesto and this is serious for sure.

The usual interpretation only addresses the first value of the manifesto ("well-crafted software"). The other three values are not addressed at all.

- The second value (*"steadily adding value"*) is completely left out because value is defined by the customer, not the developer. If we only focus on coding-related topics we cannot know if we add value or not. We do not learn anything about customer value by doing coding katas.
- Also the other two values (*"community of professionals"* and *"productive partnerships"*) are not addressed: You will get the latter only if you can talk at eye level with your stakeholders, which you do not learn by doing coding katas.
- And you will be perceived as "professional" only, if you are capable of understanding their needs and pains and adding your share to the solution. This also requires a lot more than just improving coding skills.

Without denying the need for better coding skills, focussing only on improving those skills violates three out of four values of the manifesto. Sometimes I get the feeling that the term "Software Craftsmanship" is misused by some people as a convenient legitimation to focus on the activities they like most (writing code) and to put aside the activities they do not like so much (all the other activities belonging to professional software development).

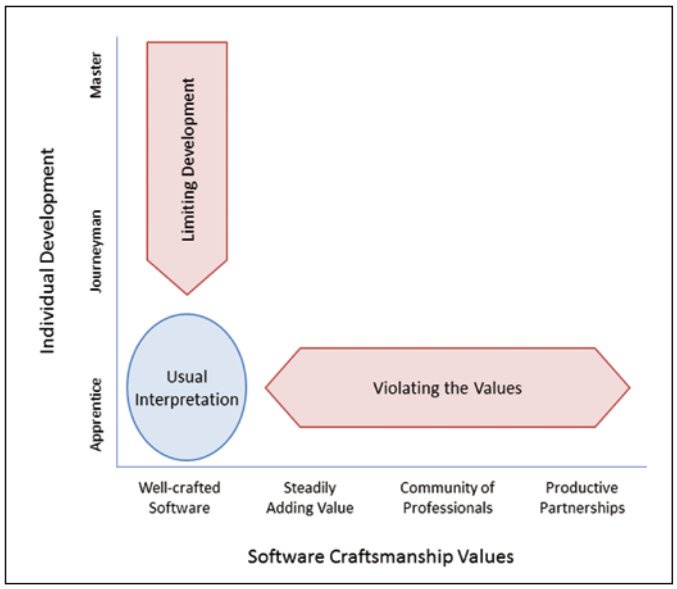


Figure 1: Limitations of the usual interpretation of Software Craftsmanship

Also I am afraid, if we as a community do not manage to balance all four values reasonably, Software Craftsmanship might get the reputation of a lame excuse for people who do not care about anything but hacking code. Probably I am a bit too pessimistic about this but focussing on coding skills only definitely is not enough.

Doing it better

Summing up, the usual interpretation as described above has two limitations:

- It limits individual development. The apprentice level is not left. A development to a journeyman or master level does not take place.
- The values of the manifesto are violated. Only the first value is considered, the other three values are neglected.

This raises the question: How should we do it better?

I think there is a place for the narrow interpretation, but it needs to be embedded into a broader context. Improving coding skills is a valuable starting point in the journey towards a software craftsman. But as personal experience grows, it becomes necessary to build up expertise beyond the pure developer perspective to become a real craftsman.

Or at greater length, craftsmanship means steadily improving the individual competencies with the purpose to create added value and quality for our customers. Our improvement activities are not guided by our personal preferences but by this purpose.

Being a junior developer this means, I have to start building up expertise in a well-defined area first because I cannot learn everything at once. Since working code is a core result of our activities it makes sense to start with improving my personal coding skills.

But with more experience I need to broaden my expertise, because coding is just one out of a series of skills I need to create added value and quality for my customers.

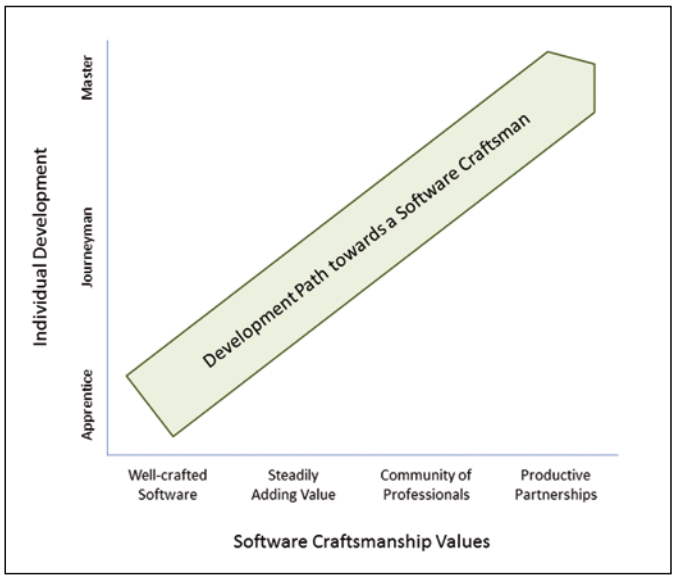


Figure 1: The path to becoming a real Software Craftsman

- For instance, a good craftsman needs to respond to his customers. Therefore he needs to learn their language to be able to communicate productively with them (and to become accepted by them).
- He needs to understand the needs and pains of his customers and he also needs to carve out those requirements a customer is not capable to couch in words easily.
- He needs to balance competing forces and to identify good compromises. He needs to make transparent the trade-offs of the alternative solution approaches and to support the customer to find the solution that matches the customer's priorities best.
- He needs to understand the environment the product is embedded in and to deal with the constraints and conflicts which may arise from it.

And so on – again, this needs a lot more than just excellent coding skills.

Conclusion

After all, you can compare a software craftsman to a trade craftsman. A trade craftsman master does a lot more than just standing in his workshop all day crafting products. He takes care about dates and money, resolves problems, coordinates suppliers and customers and does a lot more.

Likewise, a software craftsmanship master needs to take care about a lot more than just making sure the product he creates meets his personal quality standards – and he needs to acquire the competencies required for that.

This is not possible if we reduce Software Craftsmanship to Coding Katas, Code Retreats and other code-only-related topics. This might suffice at the start of our craftsmanship career. But as we grow we need to leave the comfy developer chair and go on a journey as a trade journeyman does: We need to learn about other departments and domains, we need to learn about the needs and specialities of our customers and partners and we need to improve our soft skills..

Of course it is important not to forget our coding skills on that journey and we might want to do some coding kata or code retreat on our way. But as true software craftsmen we know a lot more than just how to write nice code.

Uwe Friedrichsen has a long track record as architect, developer and project manager. His focus areas are architecture, agile software development and scalable systems. Besides writing articles and sharing his ideas on conferences he is also one of the founders of a local software craftsmanship community in Düsseldorf, Germany.

References

[1] http://manifesto.softwarecraftsmanship.org/

[2] http://agilemanifesto.org/

[3] If you do not know what those terms mean, I definitely recommend to browse the web or to join a local craftsmanship community nearby in order to learn about those things. It is your personal decision if you want to stick to the concepts those terms represent but as a good software developer at least you should know what those terms (and some more) mean.

Imprint

Publisher
Software & Support Media GmbH

Editorial Office Address
Darmstädter Landstraße 108
60598 Frankfurt am Main
Germany
www.jaxenter.com

Editor in Chief: Sebastian Meyen
Editors: Chris Mayer
Authors: Jamie Allen, David Blevins, Benjamin Cabé, Carl Dea, Uwe Friedrichsen
Copy Editor: Nicole Bechtel, Lisa Pychlau
Creative Director: Jens Mainz
Layout: Flora Feher, Maria Rudi, Petra Rüth

Sales:
Ellen May
+44 (0)20 7401 4837
ellenm@sandsmedia.com