From:	Brian Swetland.	Sent:1/3/2006 1:31 PM.
To: [-]	Mathias Agopian.	
Cc: [-]	fadden@google.com; arubin@google.com; joeo@google.com.	
Bcc: [-]	.	
Subject:	Re: new java world.	

[Mathias Agopian <mathias@google.com>]
> Has this decision been taken already or are we talking/arguing about it?

I think we're pretty set on it, but are still working on addressing
issues people may have with it. The skia folks are being brought up
to speed today. I unfortunately misremembered when you were going to
be back (thought it was the beginning not the end of this week) and
thought you would be around today to discuss things.

Brian

> On Jan 2, 2006, at 11:07 PM, Brian Swetland wrote:
>
> >
> > >Reasons to shift to a primarily Java API
> > >
> > >- single language massively simplifies the application development
> > > story: "you write android apps in java. native code is brought in
> > > as standalone modules (services) or as plugins to the runtime
> > > (components)"
> > >
> > >- single language approach massively simplifies system development
> > > and reduces our development time. The universal multilanguage
> > > binding stuff is an awesome idea but is a crazy pile of work and
> > > risky.
> > >
> > >- the tools story is much, much simpler. supporting gcc/etc cross
> > > compilers and other tools is a big pain (we have to do it for
> > > systems development but we can avoid passing that pain on to
> > > 99% of application developers).
> > >
> > >- even on a system with processes / mmu / etc, java provides a nice
> > > safetynet and faster app development and debuggability. (this
> > > is based on experience developing hiptop -- java saved us a
> > > pretty crazy amount of time).
> > >
> > >- the negotiations with Sun are going far better than expected.
> > > A lot of the push for multi-language bindings was a result of
> > > Brian trying to work out a cover-our-ass setup for when Sun
> > > proves impossible to work with (he was perhaps a bit scarred
> > > by his danger experience).
> > >
> > >- using java simplifies the "why did you invent a new api" story.
> > > The embedded java world has midp which we will have for
> > >compatibility
> > > with 'legacy' phone apps, but not much in the way of more powerful
> > > environments. We can fill this gap *and* avoid a lot of the
> > > "why didn't you just use gtk / qtopia / etc" questions.
> > >
> > >- using java allows us to take advantage of a modern, garbage
> > > collected memory model for applications without having to worry
> > > about integration with C++ allocation, reference counting, etc.
> > > The same goes for synchronization.
> > >
> > >- having one primary language environment allows us to focus all our

> > language systems development energy on making the java world
> >extremely
> > fast and solid.
> >
> >One language is okay, but why Java instead of C++, Intercal, etc?
> >
> >- The nature of the cellular market is that we are *required* to have
> > java due to carrier requirements, etc. Since we're sorta stick with
> > that, we can provide two environments (java and native) like
> >everyone
> > else does, expending a lot more energy, or simplify and just provide
> > a fantastic java environment.
> >
> >- Java is more accessable than C++. There are more Java programmers.
> > There is more standardization in tools and libraries. Debugging is
> > much simpler (especially for people who are not total rockstars --
> > perhaps a lot of casual developers, etc).
> >
> >- Java solves a lot of the portability issues C++ has. There is
> > no fragile base class problem in the sense that it exists in C++.
> > We can safely provide a modern object oriented api to third
> > party developers without the scary ABI issues involved in C++.
> > (exceptions are zero runtime cost if not throw, by design, in
> > java. garbage collection is builtin and standard. etc. etc)
> >
> >- Performance concerns? Yup, we need to do work to push the
> > heavy lifting to native (but we're already doing that!) and
> > have a care for performance and writing more of the system in
> > java does make that a little harder. We solve this by making
> > the java runtime very fast, moving what needs to be native native,
> > and being smart about writing our java code. The folks from
> > danger can explain this at length -- shipping a *fast* java based
> > system is totally doable, even on much slower hardware than we have.
> >
> >- Java does have a big win of being much more compact code than
> > native arm/thumb code.
> >
> >What changes (not all that much):
> >
> >- The biggest change is the view system becomes a java library.
> > This does involve migrating some already written code, but in the
> > end makes for much easier to use java apis if we don't restrict
> > the design to the intersection of java and c++ features.
> >
> >- Most of the low level, performance intense stuff (image libs,
> > skia, audio engine, etc) stays just as it is and gets java
> > wrappers as previously planned.
> >
> >- The browser integration mostly moves ahead as planned. It is
> > a native component represented in a java widget/view/whatever.
> > A way of having it setup and control other widgets on top of
> > (inside of?) itself will be required for forms support, etc.
> >
> >- Joe's IDL tool is still used to generate java bindings for native
> > components, but it lives in a simpler world. It is also used
> > for building c++ and java interfaces to "services" which are
> > the bigger building blocks of the system (data storage, addressbook,
> > telephony, etc)
> >
> >Components vs Services:
> >
> >- Components plug in to the core system libraries and are exposed

> > to the java world as java classes (often wrapping native code).
> > they run inside the process space of applications
> >
> >- Services run in their own process space (though one process
> > can host a number of services that are related / can share the
> > same security boundaries). Services communicate via IPC (using
> > shared memory where appropriate) and their interfaces are defined
> > by and wrappers are generated by the IDL tool.