Jakarta EE 11 Discussion

CDI Centric

https://github.com/eclipse-ee4j/jakartaee-platform/issues/552

[Arjan] Making CDI the core component model. We've been on this path ever since CDI was introduced in EE 6. Particularly in EE 7 where @Transactional was introduced, EE 8 where Security was introduced that was fully based on CDI from the get go, and EE 10 where Faces pruned its native managed beans in favour of CDI (a plan that was announced all the way back in 2009).

For instance, in Jakarta Persistence, the EntityManager isn't directly injected via CDI. See https://github.com/jakartaee/persistence/issues/377

Also, for what should be a relatively modern specification, Jakarta Concurrency is still deeply rooted in JNDI, resource-refs, EJB,etc. We should really base this more on CDI. See https://github.com/jakartaee/concurrency/issues/229 and https://github.com/jakartaee/concurrency/issues/252

Make the extension beans in other specs such as Transaction: See e.g. Jakarta Transactions: https://www.eclipse.org/lists/jta-dev/msg00245.html

[Ondro] I'm missing alternatives for some EJB constructs. Mostly in concurrency, like locking access to singletons, pooling, annotation-based timers, etc.

CDI replacing Managed Beans

Managed Beans was an attempt of a universal minimal component model, introduced in Java EE 6 and removed again in Jakarta EE 10. It specified a minimum of services, like @Inject support and @PostConstruct support.

Servlets are one example of adhering to the Managed Beans spec (without explicitly referencing the spec); a servlet supports @Inject and @PostConstruct, but it's not a CDI bean.

The proposal is to upgrade Servlets to being true CDI beans, but only in a Jakarta EE environment. This has been discussed on the servlet mailing list before, but the discussions have mostly gone nowhere. This is mostly since Greg from Jetty doesn't know what CDI is (and reasonably, we can't expect him to learn this, as it's not in Jetty's interest).

CDI replacing EJB

[Reza] In terms of remaining gaps, I think the themes that emerge are advancing CDI adoption throughout the platform, deprecating/replacing older technologies like EJB as well as introducing cloud native technologies into (new) specifications.

[Emily] We need to work out the gaps to see what else needs to move from EJB to other specs in order to deprecate EJB.

[Arjan] Some ideas being written down here:
https://omnifish.ee/2022/11/01/ejb-support-in-piranha-via-cdi/

[Ed-Bratt] Replacement or alternative for MDB

[Lenny Primak] Replace Remote EJBs with CDI RPC spec (maybe gRPC as a transport?)

[Angelo Rubini] Hi All, i think the Remote EJB Component Model have more features vs CDI bean.

In the EJB 1.1 Specification, Sun Microsystem in according to OMG integrate RMI over CORBA IIOP(Standard deFacto and more use in the years 80/90).
Most corba client invoke Ejb method from c++, ada etc by Corba Client.
Why not reproduce same scenario for HTTP2/gRPC hera?
Http2 has more fearture like Corba TCP Connection(Single Connection,Multiplezing, priority).
The good base of RMI-IIOP based to OMG Corba(IDL+IIOP/GIOP Message) have most power.
gRPC or Finangle sponsored an rpc over http2(only efficient alternative for REST JSON+HTTP), but there are more work for developer.

- generate proto interface(on java there is java interface by OMG IDL),
- generate and populate stub code(on Ejb this is gratis with Runtime Stub Generation/Downloading feature).

generate and populate skeleton code(on Ejb this is gratis with Runtime Skeleton Generation feature).
for a server-to-server communication is a good choice ejb-to-ejb communication, and support distributed transaction managed by conintainer.
The EJB CMT Feature (EJB container managed transaction) is power feature.
An interesting project is jakarta-RPC but in this moment is very distance from EJB.
Others solutions like:
Wildfly:
https://github.com/wildfly/wildfly-http-client/blob/master/docs/src/main/asciidoc/wire-spec-v1.asciidoc
Payara: https://blog.payara.fish/remote-ejb-via-http
TomEE: https://tomee.apache.org/ejb-over-ssl.html
weblogic: offer http tunneling for t3 protocol..

[Petr Aubrecht] Is there a single annotation in CDI equal to Stateless in EJB?

Core HTTP support

Currently we have two distinct HTTP stacks in Jakarta EE: Jakarta REST, which is in Core Profile, and Servlet.

Although practically a lot of Jakarta REST implementations are based on Servlet, this is officially not the case. As such, Servlet is not in the Core Profile, making Jakarta REST more “core” than Servlet.

Other specs that build on HTTP support, such as Jakarta Security and Jakarta Faces, build on Servlet. That makes Servlet more “core”.

Obviously having two overlapping cores is not ideal, and we should probably split out an actual core on which both Jakarta REST and Jakarta Servlet can build.

[Markus] The question is, what the actual benefit for the application programmer would be? JAX-RS has no problem, Servlet has no problem. So this sounds like a clean solution but with no added value. JAX-RS is in the core, because people like its simplicity. Servlet is not in the core, because least people need it, as they already have JAX-RS.

[Arjan] The question is, what the actual benefit for the application programmer would be?

-> A consistent API. Just as we don’t want 10 ways to inject something, we should not have multiple ways to get a header from a request.

> REST has no problem

REST has a big problem

> Servlet has no problem.

Servlet has a big problem

MicroProfile and Jakarta Alignment

[Ondro] In general, pulling some features from Microprofile that fit Jakarta EE would be useful, e.g. Rest client, which nicely fits into Jakarta REST already.

[Arjan] MP JWT makes a lot of sense as an additional authentication mechanism for Jakarta Security. In practice, a variety of implementations already use it that way. It would be interesting how we would exactly do this. Copy over the spec text into Jakarta EE? Reference a specific version? E.g. Security 3.1 saying under "provided authentication mechanisms" something like:

"Support for the JWT authentication mechanism is required, as specified by the Microprofile JWT spec, with the following additions/changes..."

Complete adding MP Context Propagation content into Concurrency

Also, if MP needs a feature add those features to Core Profile (ex Bean Validation, Concurrency)

Possible improvements in Jakarta specs that may help move MicroProfile specs forward

[John] There are non-technical considerations regarding alignment, like how the two working groups operate and the membership cost.

CORS Support

[Vedran] @CrossOrigin - configurable CORS annotation. Should support Rest and Servlet based methods and classes. Should also support something like apply to everything, maybe even EL etc. I know it is easy to write a custom filter but i think a standard way for doing this would be nice.
[Hantsy] Hope it is part of security. we also need to consider general CSRF(not just put into a form in Faces/MVC) and other items listed in https://owasp.org/www-project-top-ten/

Jakarta Config

[Ondro] it would be nice to have Jakarta Config in EE 11 and some other specifications using it for configuration. In general, pulling some features from Microprofile that fit Jakarta EE would be useful, e.g. Rest client, which nicely fits into Jakarta REST already.

Config working on a MVP. Maybe release a 1.0 before EE 11?

Adopt Java SE 11, 17, 21 new features and Breaking Changes

https://github.com/eclipse-ee4j/jakartaee-platform/issues/553

[Ondro] Adopting new features in Java 11 and 17 is also something that people expect these days. Mostly records, but I think support for some older constructs like the reactive Flow API are still missing.

[Hantsy] Add Flow to all async case, such as return type(Flow.Publisher, such as the implementations Smallrye Mutiny Muti/Uni) of Concurrency, also consider add to CDI async event, JMS async eventlistener.
Smallrye Mutiny is switching to use Flow instead of reactivestreams.

Handle removal of Java SecurityManager and the impact that it has to the Jakarta Security specification

Limit the number of SE versions to be required to support (run the TCK with)

Right now some TCKs will not run on Java levels above Java 17. This should be made a lot easier and not require so much Java level specific code. Signature tests in some TCKs is an example where this is a problem.

What is the gap between 11 and 21 that we need to consider?
Language adoption vs minimum required Java SE version and maximum supported Java SE version

Which language features should be considered?
Call to action for the specification projects for what Java SE features (after 11) that they want to incorporate into their specification

Explore the support of Project Loom - virtual thread

[Emily] Since the project loom is in Java 19, Jakarta EE Concurrency should investigate how to embrace virtual thread and prepare to work with Java 21.

Other features

[Vedran] Interceptor support in EL

Standardizing the way Jakarta apps are tested

Move JCache to Jakarta EE?

TCK Refactoring

https://github.com/eclipse-ee4j/jakartaee-platform/issues/554

Continue decoupling

Continue modernizing

Making standalone TCKs running easily in the EE context
Standardizing how TCKs are run, configured, and exclusions supported

[Jan] Creating a (component) spec template as showcase might be helpful and this could act as a testbed for a parent project like we have in microprofile-parent for offering default Maven execution too.

Test framework

[Emily] How to test Jakarta EE applications? Maybe create a library etc.

Jakarta Commons Project

This was talked about at one point, was it created?

Continuous Integration

[Jan] Using the Jakarta TCKs for continuous integration testing (on every change), instead of validating Compatible Implementations (mostly very late in the process) only.

This would require some efforts, like:

- Splitting the TCK up to the component specs
- Establishing and maintaining naming conventions