Meecrowave JPA

The overall idea behind this module is to propose a CDI integration of JPA allowing to programmatically control its persistence units.

Concretely you will create a persistence unit from a `PersistenceUnitBuilder` allowing you to fully configure your unit from CDI context including the datasource:

```
@ApplicationScoped
public class JpaConfig {
    @Produces
    public PersistenceUnitInfoBuilder unit(final DataSource ds) {
        return new PersistenceUnitInfoBuilder()
                .setUnitName("test")
                .setDataSource(ds)
                .setExcludeUnlistedClasses(true)
                .addManagedClazz(User.class)
                .addProperty("openjpa.RuntimeUnenhancedClasses", "supported")
                .addProperty("openjpa.jdbc.SynchronizeMappings", "buildSchema");
    }
}
```

if your application uses a single persistence unit this is optional and a default one will be created if a single DataSource bean is available as Bean<?>.

The datasource can be produces as you wish using your own configuration mecanism:

```
@ApplicationScoped
public class JpaConfig {
    @Produces // dbcp2 datasource for instance
    @ApplicationScoped
    public DataSource dataSource() {
        final BasicDataSource source = new BasicDataSource();
        source.setDriver(new Driver());
        source.setUrl("jdbc:h2:mem:jpaextensiontest");
        return source;
    }
}
```

it is recommanded to ensure the `DataSource` is normal-scoped to not get surprises in term of behavior.

Finally you can inject your entity manager using `@Unit`. Ensure to decorate with `@Jpa` a class/method before using the entity manager to activate the jpa CDI context:

```
@ApplicationScoped
@Jpa(transactional = false)
public class JPADao {
    @Inject
    @Unit(name = "test")
    private EntityManager em;

    @Jpa // with a resource local transaction
    public User save(final User user) {
        em.persist(user);
        return user;
    }

    // inherit form class, no tx
    public User find(final long id) {
        return em.find(User.class, id);
    }
}
```

this integration is 100% based on `RESOURCE_LOCAL` units for now.

Not that if a bean get injected an `EntityManager` it gets automatically `@Jpa(transactional=true)` so previous bean is equivalent to:

```
@ApplicationScoped
public class JPADao {
    @Inject
    @Unit(name = "test")
    private EntityManager em;

    public User save(final User user) {
        em.persist(user);
        return user;
    }

    @Jpa(transactional = false)
    public User find(final long id) {
        return em.find(User.class, id);
    }
}
```

Integration with Bean Validation

The extension will try to find a `ValidatorFactory` in CDI context and will provide ir to the JPA provider if the `ValidationMode` is not `NONE` and a `Bean<ValidatorFactory>` exists.