

School of Computer Science
Language Technologies Institute

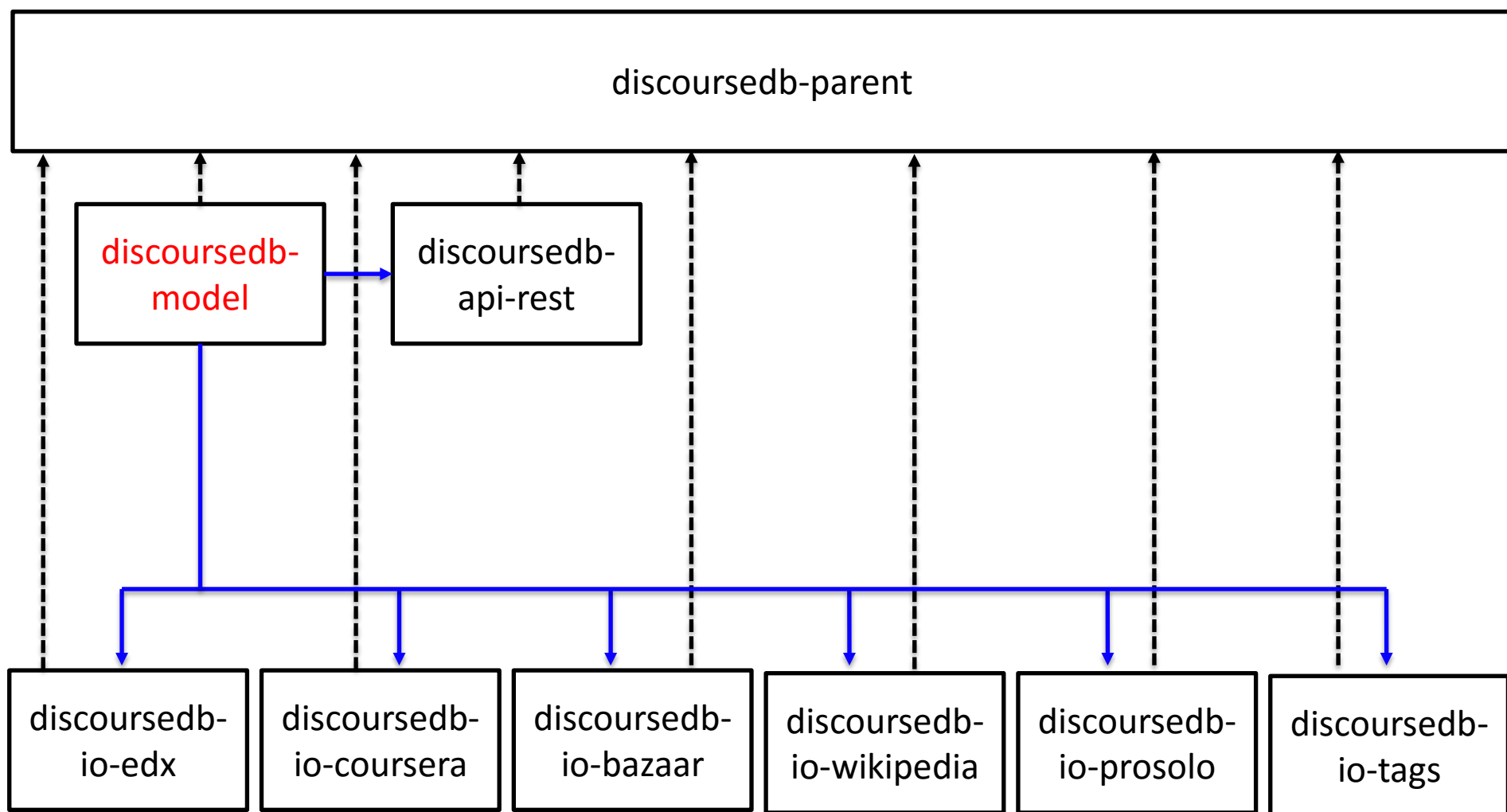
DiscourseDB

Project Structure

Maven

- DiscourseDB is a multi-module Maven Project
- Main project properties and central dependency management located in the parent project
- Continuously built by Jenkins build server
- Successful builds automatically deployed to Artifactory

Project Layout



Parent

The screenshot displays the IntelliJ IDEA IDE interface for a Maven project. The left pane shows the 'Overview' tab with the following details:

- Artifact:**
 - Group Id: edu.cmu.cs.lti
 - Artifact Id: discoursedb-parent
 - Version: 0.5-SNAPSHOT
 - Packaging: pom
- Parent:** (empty)
- Properties:**
 - project.build.sourceEncoding : UTF-8
 - maven.compiler.target : 1.8
 - maven.compiler.source : 1.8
 - maven.surefire.heap : 512m
- Modules:**
 - ../discoursedb-io-edx
 - ../discoursedb-io-bazaar
 - ../discoursedb-io-tags
 - ../discoursedb-io-prosolo
 - ../discoursedb-model
 - ../discoursedb-api-rest
 - ../discoursedb-io-wikipedia
 - ../discoursedb-io-coursera

The right pane shows the 'Dependencies' tab, which is divided into two sections:

- Dependencies:** Lists managed dependencies like log4j-api (managed:2.4.1) and log4j-core (managed:2.4.1).
- Dependency Management:** Lists various dependencies with their versions and scopes, such as platform-bom (2.0.0.RELEASE), mysql-connector-java (5.1.38), and various querydsl and jackson dependencies.

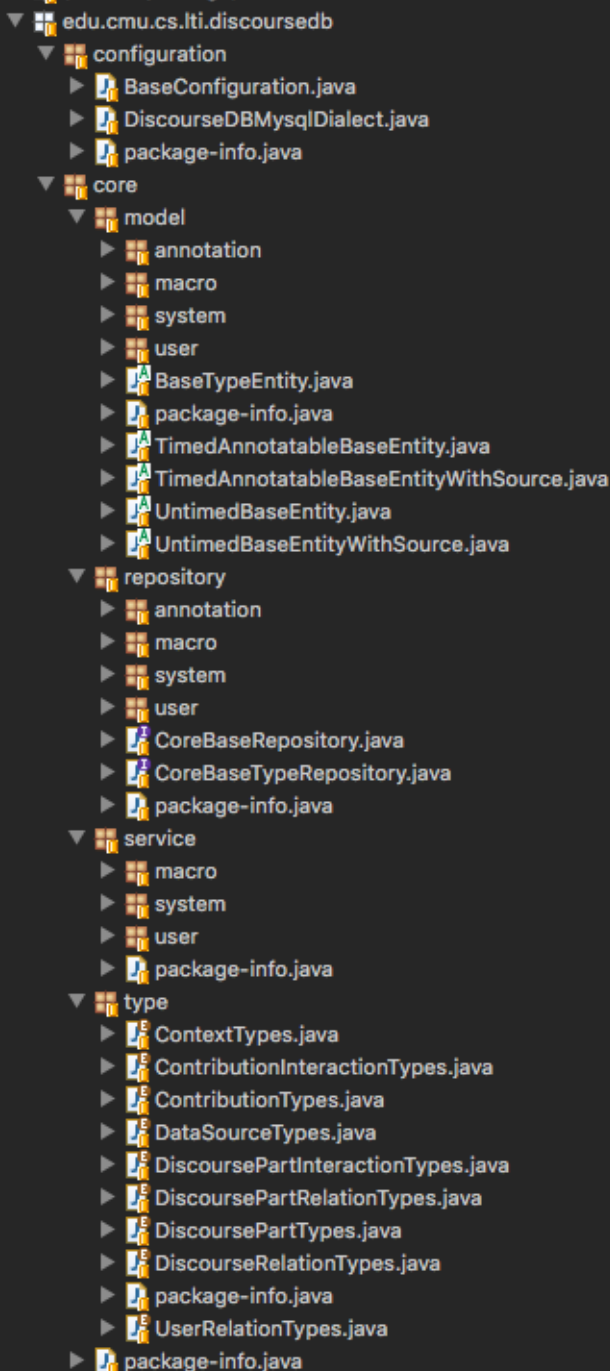
- Define library versions to be used by child projects
- Set up project layout

discoursedb-model

- The core module represents
 - The DiscourseDB data model
 - Data Access Layer
 - Default configuration

Model Layout

- Model
 - Persistence entities with ORM annotations
- Repository
 - Spring Data repository classes for the core entity beans that provide low-level data access methods
- Service
 - Service-layer classes which use repositories to provide high-level data access methods
- Type
 - Type definition for DiscourseDB type entites



Persistence Entities

- POJO that represents persistent data maintained in database
- Similar concept as EJB Entity Beans
- Instances of such an entity correspond to individual rows in the corresponding table
- Entities have relationships with other entities: expressed through object/relational metadata → annotations

Persistence Entities Example

```
@Entity
@Table(name="content")
public class Content extends TimedAnnotatableBaseEntityWithSource implements Serializable {

    private static final long serialVersionUID = -1465025480150664388L;

    private long id;

    private Content previousRevision;

    private Content nextRevision;

    private String title;

    private String text;

    private Blob data;

    private User author;

    private Set<ContributionInteraction> contributionInteractions = new HashSet<ContributionInteraction>();

    public Content() {}

    @OneToOne(cascade=CascadeType.ALL)
    @JoinColumn(name = "fk_user_id")
    public User getAuthor() {
        return author;
    }

    public void setAuthor(User author) {
        this.author = author;
    }

    @Id
    @Column(name="id_content", nullable=false)
    @GeneratedValue(strategy = GenerationType.AUTO)
    public long getId() {
        return id;
    }
}
```


Spring Data Repositories

- Reduce the amount of boilerplate code required to implement data access layers for various persistence stores.
 - ➔ Avoid the need to write code that creates database queries
- Define repository interfaces without worrying about their implementation

Encode Query in method names

- Define methods in repository interface
- Let Spring Data implement the methods on the fly

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true

Examples

```
public interface DiscoursePartRepository extends CoreBaseRepository<DiscoursePart, Long>{  
    Optional<DiscoursePart> findOneByName(String name);  
    List<DiscoursePart> findAllByName(String name);  
    List<DiscoursePart> findAllByType(DiscoursePartType type);  
}
```

```
public interface UserRelationRepository extends CoreBaseRepository<UserRelation, Long>{  
    Optional<UserRelation> findOneBySourceAndTargetAndType(User source, User target, UserRelationType type);  
}
```

```
public interface ContentRepository extends CoreBaseRepository<Content, Long> {  
    public List<Content> findByIdIn(List<Long> contentIdList);  
  
    @Modifying  
    @Query(value = "update content c set c.fk_next_revision = ?2 where c.id_content = ?1", nativeQuery = true)  
    public void setNextRevisionId(Long id, Long nextRevId);  
  
    @Modifying  
    @Query(value = "update content c set c.fk_previous_revision = ?2 where c.id_content = ?1", nativeQuery = true)  
    public void setPreviousRevisionId(Long id, Long previousRevId);  
}
```

Basic CRUD capabilities

- Base interface provides low level access capabilities to all entities

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    <S extends T> S save(S entity);

    T findOne(ID primaryKey);

    Iterable<T> findAll();

    Long count();

    void delete(T entity);

    boolean exists(ID primaryKey);

    // ... more functionality omitted.
}
```

The Service Layer

- provides a higher level of abstraction for data access.
- services encapsulate whole processes and allow to perform additional consistency and validity checks
- **repositories** define access methods for single entities while **services** can interact with multiple entities
- services use repositories (and potentially also other services)

Service Examples

```
@Transactional(propagation= Propagation.REQUIRED, readOnly=false)
@Service
public class ContributionService {

    @Autowired private ContributionRepository contributionRepo;
    @Autowired private DataSourceService dataSourceService;
    @Autowired private ContributionTypeRepository contribTypeRepo;
    @Autowired private DiscourseRelationTypeRepository discRelationTypeRepo;
    @Autowired private DiscourseRelationRepository discourseRelationRepo;

    /**
     * Retrieves existing or creates a new ContributionType entity with the
     * provided type. It then creates a new empty Contribution entity and
     * connects it with the type. Both changed/created entities are saved to
     * DiscourseDB and the empty typed Contribution is returned. It then adds
     * the new empty Contribution to the db and returns the object.
     *
     * @param type
     *      the value for the ContributionType
     * @return a new empty Contribution that is already saved to the db and
     *      connected with its requested type
     */
    public Contribution createTypedContribution(ContributionTypes type){
        Assert.notNull(type);
        Optional<ContributionType> optContribType = contribTypeRepo.findOneByType(type.name());
        ContributionType contribType = null;
        if(optContribType.isPresent()){
            contribType = optContribType.get();
        }else{
            contribType = new ContributionType();
            contribType.setType(type.name());
            contribType = contribTypeRepo.save(contribType);
        }

        Contribution contrib = new Contribution();
        contrib.setType(contribType);
        return contributionRepo.save(contrib);
    }
}
```

QueryDSL

- It is very hard to implement complex queries and even harder to read them.
- Defining repository queries is fast and easy for single entities, but verbose
- If joins are involved, repository-style queries are not ideal
- QueryDSL
 - abstraction layer for queries
 - allows to define reusable predicates that can be passed to repository methods

QueryDSL Example

```
@Service
public class ExampleService{

    @Autowired
    private UserRepository userRepository;

    public Iterable<User> findUsersBySourceId(String sourceId) {
        return userRepo.findAll(
            QUser.user.dataSourceAggregate.sources.any().entitySourceId.eq(sourceId)
        );
    }
}
```

- **Retrieves all User entities that have an associated DataSourceInstance which contains the provided sourceId**
- The QUser class is autogenerated by QueryDSL
- Predicates (the argument of the findAll() method) can be stores in a separate Predicate class so it can be re-used in multiple queries