Introduction to the Business Objects 2 framework (Bo2)

INTERAMERICAN
PART OF ACHMEA

S. Nakos, D. Kateros

INTERAMERICAN
PART OF ACHMEA

The basics

INTERAMERICAN
PART OF ACHMEA

Principles

- Reusability
- Long term maintenability
- Service orientation
- Model driven development
- Agility

INTERAMERICAN
PART OF ACHMEA

Applicability and scope

- General purpose business layer framework written in Java 1.6.
- Bo2 defines a number of core concepts (abstractions) that can be used by a development team in order to tackle business tasks in a semantically uniform manner. The developers are hindered from using custom solutions that are hard to comprehend by anyone other than the original developer.
- Instead, once a new team member has a grasp of the core concepts, we find it is very easy to read, comprehend and modify existing code.
- Implementations of these abstractions that integrate with commonly used JEE technologies (JDBC, Hibernate, Wicket, ODF) are built-in.
- Provides a large variety of general purpose java utilities as well as utilities and facades for technologies it integrates with. Bo2 strives for business code to have no compile time dependency to 3rd party frameworks.
- Built-in conventions for describing system entities with metadata and provides implementations for using the metadata in order to generalize software solutions to any system entity (e.g. rendering UI for an entity)
- Support for multiple languages.
- For the most part, the developer can concentrate on the business problem rather than solving problems relating to the enabling underlying technologies and effectively coupling business code with them.

INTERAMERICAN
PART OF ACHMEA

Core concepts

- The same classes should work on different environments
- Layered architecture
  - User interface
  - Environment: Resource management (Transactions, DB, Files, etc)
  - Business logic
  - Data access
- Defined interface for each stereotype class
- Standard checked exceptions
- Hide dependencies on third party libraries behind facades
- Method based dependency injection
- Separate interface from implementation using factories
- Eliminate boilerplate code
- Utilities
  - Metadata framework
  - Template engine based on Open Document Format
  - SQL parsing
  - Wicket
  - Bindings with Wicket, BO

INTERAMERICAN
PART OF ACHMEA

User interface

- User interface shouldn't depend on the underlying business layer implementation, but this is not always the case.
- Can be used out of the box by Java-Web architectures based on the command pattern (struts action, servlet, etc)
- Wicket (versions 1.4 and 1.5)
  - Detached lazily loaded objects
  - Auto-rendering for entities
  - Caching drop-down elements
  - Infrastructure for multilingual applications
  - Additional utilities
  - Wicket tester support
- Swing utilities in early stage

INTERAMERICAN
PART OF ACHMEA

Environment (Runtime layer)

- `Provider` handles resources lifecycle
- `ResourceWrapper` is an abstraction for managed resources. (Database connection, HibernateSession, File etc)
- `TransactionManager` handles transactions
  - JTA support inside the AS
  - JTA using JOTM outside the AS
  - DB connection based for single connection applications
- Environment variables are specified in a *deployment project.*
- `RuntimeCommand`, `BatchProcess`, and `Bo2WicketRequestCycle` bind the environment with the business layer.

```
public interface Provider {

  public <C extends ResourceWrapper>
  C getResource (String resourceName, Class<C> subclass)
  throws InitializationException;

  public TransactionManager getTransactionManager();

  public void close() throws DataException;

}
```

```
public interface HibernateSessionProvider
extends ResourceWrapper {

  public Session getHibernateSession();


}
```

INTERAMERICAN
PART OF ACHMEA

Business Layer

- Entities (Persistent objects) and Keys
- Operations
- Rules
- Utilities

INTERAMERICAN
PART OF ACHMEA

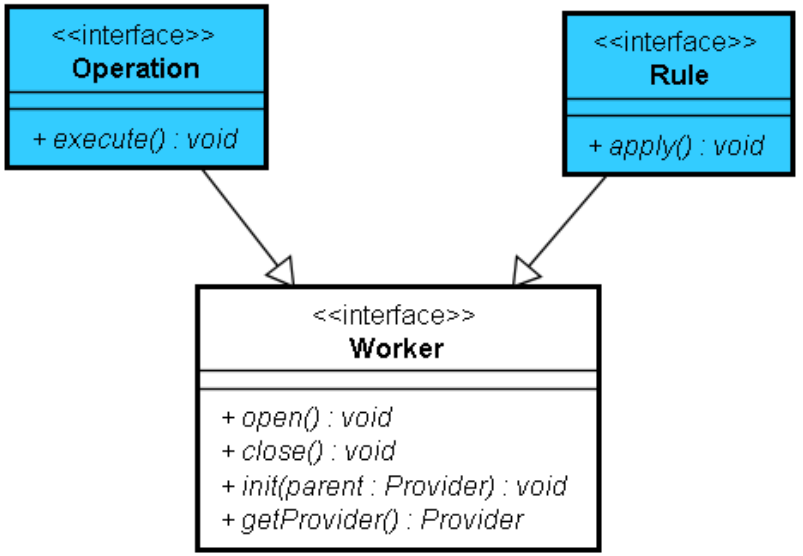
PersistentObject (PO)

- System entity that is persisted in the database.
- Has a Key. This is the database id. Object equality/hashcode/comparison is based on its key.
- Is storage agnostic. Another class handles its persistence (PersistenceWorker).
- May declare fields of Child PersistentObjects (annotated with @Child). This implies an ownership association that should be reflected on the database (foreign key constraints) and on the ORM (cascaded delete/save/update).
- Contain their own business logic, no need to be simple javabeans.
- Base implementations: AbstractBasePo, AbstractModificationRecordPo, AbstractEntryPo
- Full support of composite keys

INTERAMERICAN
PART OF ACHMEA

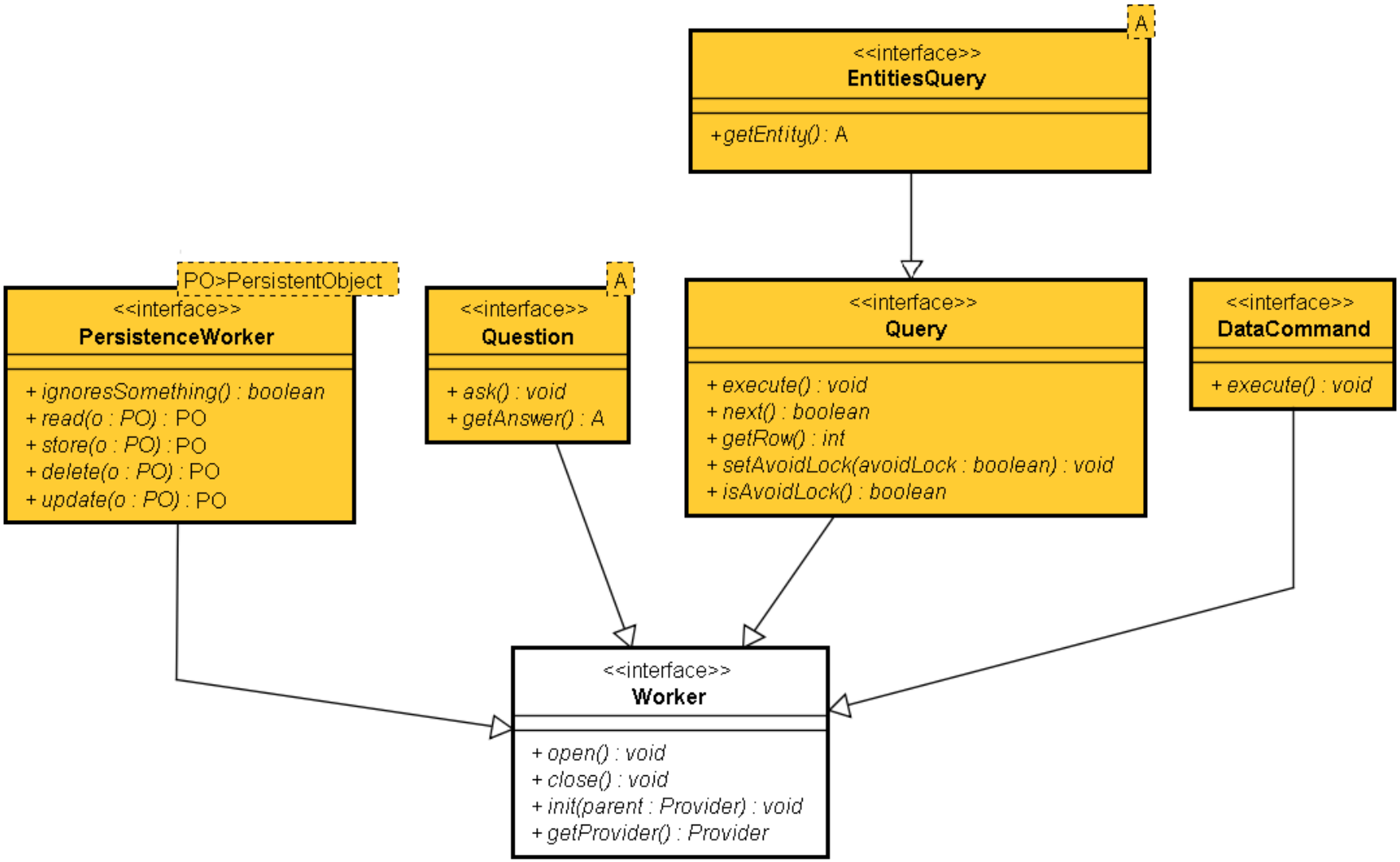
Business logic classes

- `Operation` is a worker. Performs a simple or more complex task that incorporates business logic. Accepts input using setter methods. Its `execute()` method may modify the state of the objects it operates on.
- `Rule` is a worker. Performs validations and other checks. If its `apply()` method modifies the state of the objects it operates on then this must be explained in the javadoc.
- Business logic utilities don't need managed resources. The utility class pattern proved problematic. Separation of interface and implementation should be applied also for utilities.
- Service orientation: All three stereotypes are stateless.



INTERAMERICAN
PART OF ACHMEA

Data Layer (1)



INTERAMERICAN
PART OF ACHMEA

Data layer (2)

- The CRUD methods are declared by the `PersistenceUtility` interface for any type of object. `PersistenceWorker` is a Worker and a PersistenceUtility that operates on a type of `PersistentObject`.
- The framework's Factory creates a PersistenceWorker for a Persistent class.
  - `PersistenceWorker<MyEntity> pw = Factory.createPw(MyEntity.class);`
- `Query` is an abstraction for an iterator on the data layer. Usually wraps a ResultSet, a file or a hibernate iterator. Can be used to wrap a web service or any other data layer operation that returns multiple results as an iterator. The next() method moves to the next row. Information is retrieved using an interface similar to the ResultSet with methods like `getString(columnIndex)`, `getInt(columnIndex)` etc. Column index can be int or String.
- `EntitiesQuery` returns an entity for each row.
- `Question` is an abstraction for requesting a single information that is returned as one object.
- `DataCommand` models the stereotype of a command to the data layer such as a massive update. It can hide a call to a web service, an EJB or any other external system.

INTERAMERICAN
PART OF ACHMEA

Checked exceptions

- The framework's methods throw a number of checked exceptions.
- `LogicException`: Indicates something is wrong with the actions that are being performed. Thrown by `Operation`. Also thrown by `Question` when the answer found in the data layer does not make sense from a business logic point of view.
- `RuleException`: Sub-type of `LogicException`. Indicates the failure of a validation or other type of business rule. Thrown by `Rule`.
- `DataException`: Something went wrong on an operation of the data access layer.
- `DataAccessException`: Sub-type of `DataException`. Thrown by the `Query` getObject(col) type of methods.
- `PoNotFoundException`: Is thrown by a `PersistentWorker` and indicates the requested entity is not existent in the data store.
- `InitializationException`: Something (e.g. a resource) failed to initialize properly.
- `TransactionManagerException`: Something went wrong on a transaction operation (begin, commit, rollback).
- In most cases, the developer need not perform exception handling, however he is required to be able to rethrow another exception type (should it ever come to that) as the correct Bo2 defined checked exception type.

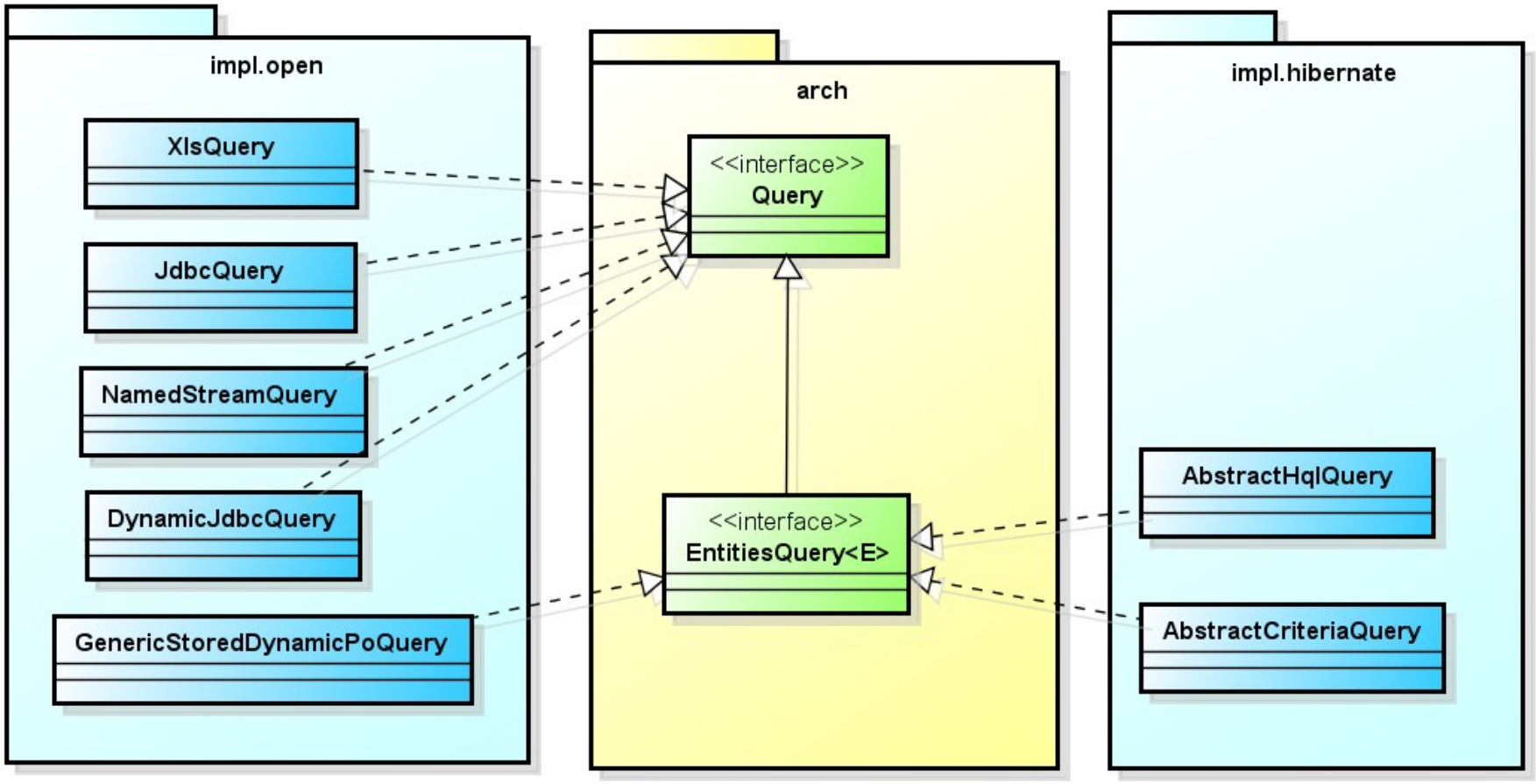
INTERAMERICAN
PART OF ACHMEA

Architecture and Implementation (1)

- All worker interfaces are declared in the Bo2Architecture project.
- Bo2ImplOpen provides implementations based on JDBC and file access.
- Bo2ImplHibernate provides implementations based on Hibernate.
- Bo2BoAdapters provides implementations based on Bo classes.
- All types of implementations can be used in the same project for the same entities.

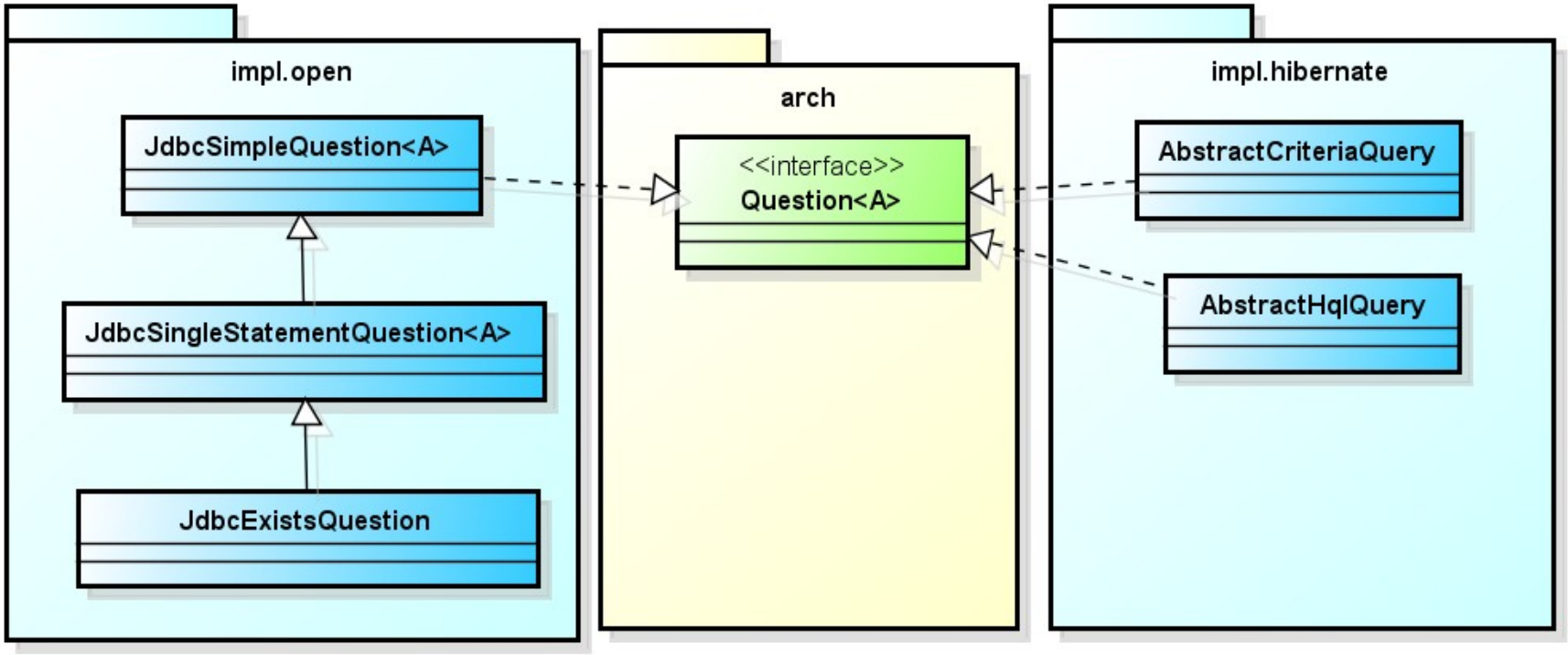
INTERAMERICAN
PART OF ACHMEA

Query types



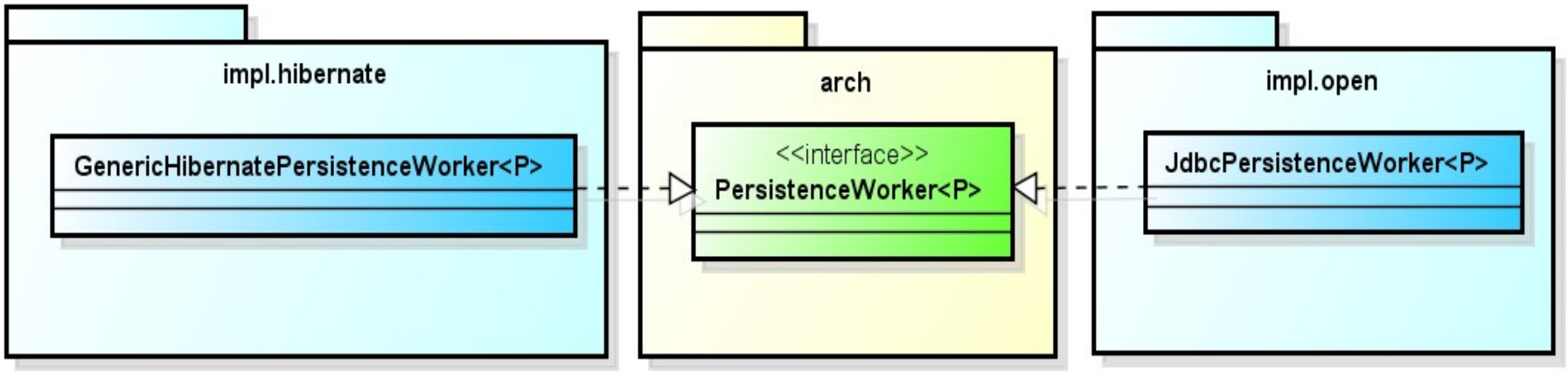
INTERAMERICAN
PART OF ACHMEA

Question types



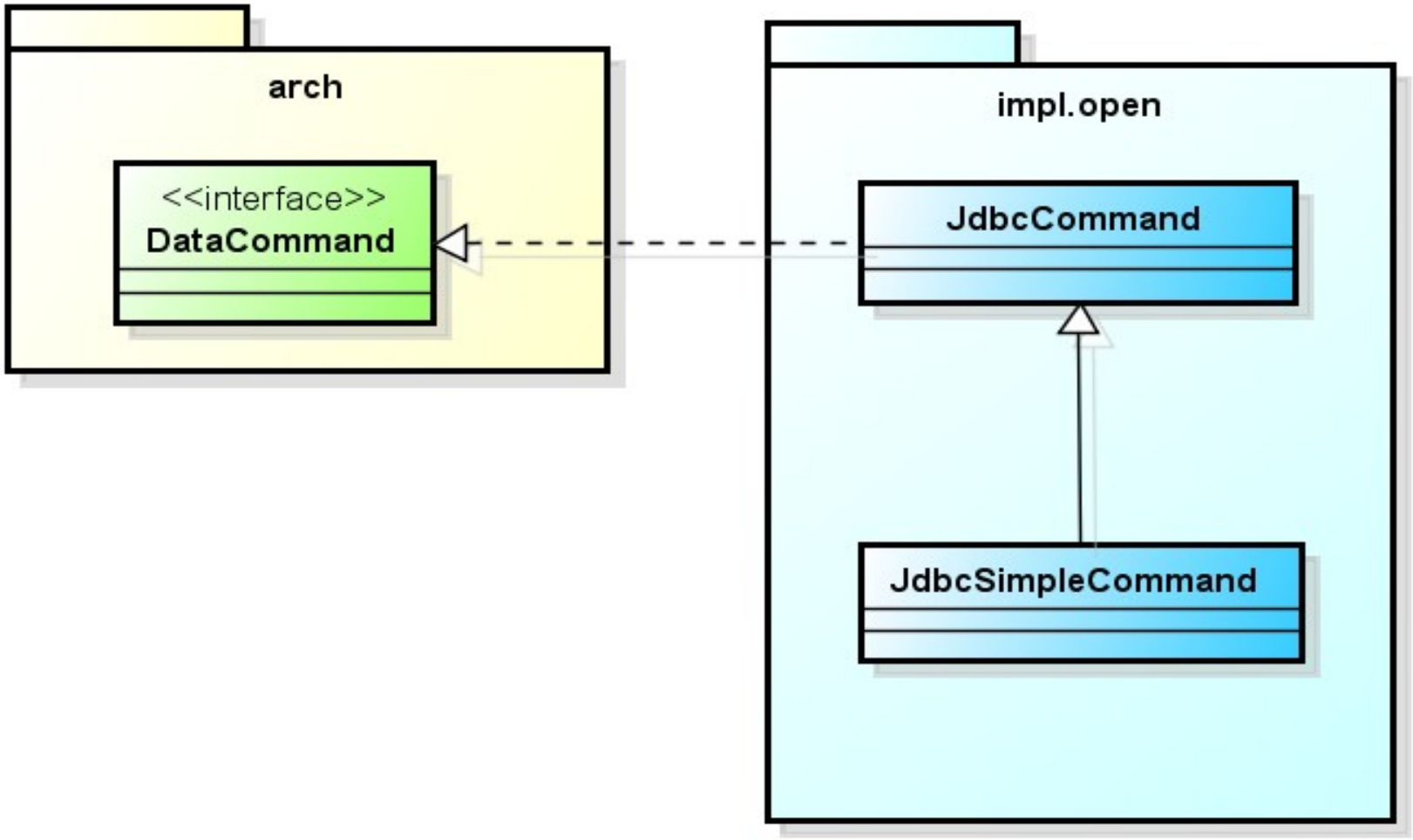
INTERAMERICAN
PART OF ACHMEA

PersistenceWorker types



INTERAMERICAN
PART OF ACHMEA

DataCommand types



INTERAMERICAN
PART OF ACHMEA

ObjectFactory

- Achieve low coupling making object creation implementation agnostic (to the source code).
- Tracks implementation – declaration associations.
- Class enhancement (less repeating code, declaratively achieve multiple inheritance and more). Leverages meta-data on implementation classes on annotations like @Property, @DelegateProperties, @DelegateMethods, @TypedSelectableProperties
- Simple JavaBeans need no implementations.
- Factory is a façade for the current ObjectFactory (the application may even switch implementations while running).
- Default declaration <-> implementation name mapping strategy, special declaration <-> implementation name mappings, as well class enhancers are configurable.
- A separate factory, PersistenceWorkerFactory creates PersistentWorkers through PersistentObject interfaces and maintains the association. It is accessible through the Factory façade as well.

INTERAMERICAN
PART OF ACHMEA

And more ...

INTERAMERICAN
PART OF ACHMEA

Parsed dynamic SQL queries

- Bo2UtilsSql defines the interfaces for an SQL parser that can be used to eliminate the code required to produce dynamically SQL queries.
- The SQL parser removes unnecessary parts of an SQL query, according to the values set on its parameters.
- Two implementation projects based on different SQL parsers:
  - Bo2JsqlParser
  - Bo2ZqlParser
- `GenericStoredDynamicEntitiesQuery`, `GenericStoredDynamicPoQuery`, `StoredDynamicPoQuery` take advantage of this functionality.

```
public abstract class ExpiringPoliciesQueryImpl
extends StoredDynamicEntitiesQuery implements ExpiringPoliciesQuery {
  static final String SQL =  "/gr/iag/queries/expiringPoliciesQuery.sql"; //Path to SQL
  public ExpiringPoliciesQueryImpl() {
     super(SQL);
  }
}
```

```
select distinct
 item.BRANCH_ID, item.POLICY_NO, item.RECEIPT_NO, item.RENEWAL_NO, item.RISK_NO, item.ITEM_NO,
 policy.TRANSACTION_RENEW, trn.TRANSACTION_TYPE, trn.AGENT_CD, item.TARIF_DT
from X__X.TB1IPOLI policy join X__X.TB1ITRAN trn on (policy.BRANCH_ID = trn.BRANCH_ID and policy.POLICY_NO = trn.POLICY_NO)
join X__X.TB1IPERI prd on (prd.BRANCH_ID = trn.BRANCH_ID and prd.POLICY_NO=trn.POLICY_NO and prd.RECEIPT_NO=trn.RECEIPT_NO)
join X__X.TB1IRISK risk on (risk.BRANCH_ID = prd.BRANCH_ID and risk.POLICY_NO=prd.POLICY_NO
                            and risk.RECEIPT_NO=prd.RECEIPT_NO and risk.RENEWAL_NO=period.RENEWAL_NO)
join X__X.TB1IINIT item on (item.BRANCH_ID=risk.BRANCH_ID and item.POLICY_NO=risk.POLICY_NO and item.RECEIPT_NO=risk.RECEIPT_NO
                            and item.RENEWAL_NO=risk.RENEWAL_NO and item.RISK_NO=risk.RISK_NO)
where policy.POLICY_STATUS = 2 and policy.POLICY_CATEGORY <> 3 and policy.EXPIRATION_DT > :startDate
and policy.EXPIRATION_DT < :endDate and trn.RENEWING_PROCESS in (2,3) and trn.AGENT_CD is  not null
```

INTERAMERICAN
PART OF ACHMEA

Metadata framework

- Defines a formalism that can describe an entity based on metadata that have been pre-defined for it.
- Automatic entity validation
- Automatic UI generation and form validation
- Cross property validation supported (e.g. property1 must be greater than property2)
- Basic classes:
  - PropertyDefinition: Holds all the data that are necessary to create a BoPropertyDescriptor: name, readOnly, hasDefault, defaultValue, zeroAllowed, negativeAllowed, nullAllowed... This is the necessary setup required.
  - BoPropertyDescriptor: Describes a business object property. Validates a value of the property based on the defined constraints. Formats the property value to a String and parses a String to a value (the transformation is symmetric).
  - BusinessObjectDescriptor: Describes a business object. Holds a reference to one BoPropertyDescriptor for each business object property. Validates a business object instance.

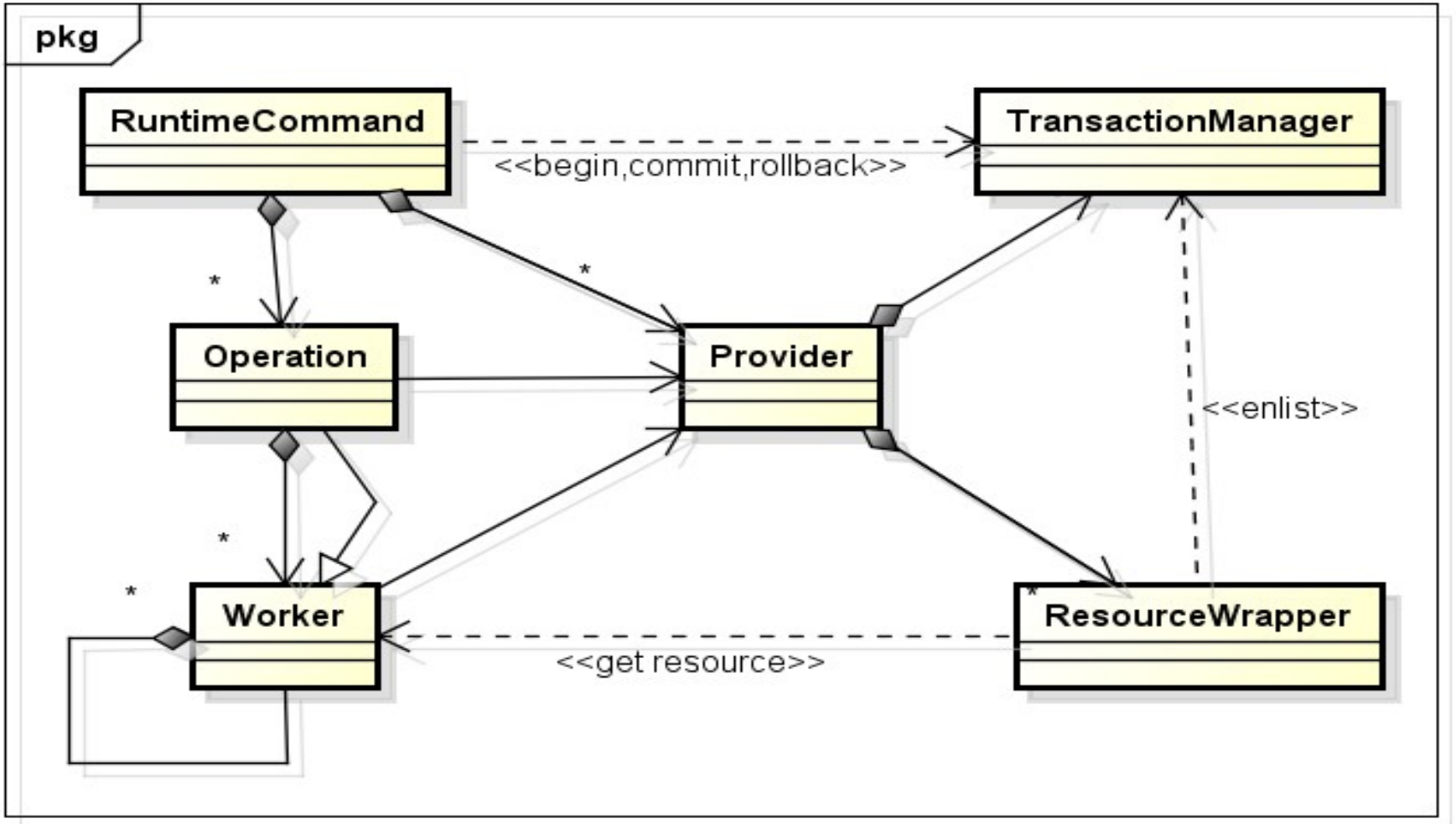
INTERAMERICAN
PART OF ACHMEA

Performing a business task (1)

- In standalone java programs transactional resources are enlisted on the TransactionManager of the current RuntimeCommand as soon as they are created. Multiple threads may work simultaneously, as the TransactionManager delegates to a JTA implementation.
- On a web environment, assuming for example instance the popular transaction per request pattern, a servlet filter may be responsible for instantiating an object that serves as a the context of a RuntimeCommand on the current Thread scoped on the current request – response cycle.
- Web components may use the ThreadLocal reference to this object in order to transparently execute Operations or use other Workers.
- Transaction management is performed by delegating to the application server's JTA implementation. Transactional resources acquired by the container are registered implicitly, as per the spec.
- The business code on the web layer should be minimal and ideally should be limited to passing input to operations and executing them or performing elementary tasks with other workers (e.g. saving an entity that was just edited in a Form with a PersistentWorker).

INTERAMERICAN
PART OF ACHMEA

Performing a business task (2)



powered by astah*

INTERAMERICAN
PART OF ACHMEA

Integration with Hibernate

- Hibernate 3.6.8
- Default PersistenceWorker implementation for any PersistentObject is Hibernate based.
- Any other implementation has to be explicitly declared in pwTypes.properties
- Hibernate mappings define interfaces. Entity keys are always composite and define a custom component tuplizer, Bo2PojoComponentTuplizer.
- Optimistic locking is performed using a <timestamp/> on property lastModified.
- On the web layer, the pattern is Session per request with detached objects. Re-attaching the working objects is the first thing to do in the beginning of a request. When the re-attaching of a persistent object is requested the framework automatically re-attaches many-to-one associations of the PO's graph as read-only. The PO itself (and all it's children are re-attached for update).
- Custom types allow the storing of Enumerations and cached entries as integers (code only).

INTERAMERICAN
PART OF ACHMEA

Integration with Wicket

- Wicket 1.4.17 and 1.5.9
- A number of pre-built components that represent commonly found use cases throughout a information system
  - Display an entity and perform some operation on it when a button is pressed.
  - CRUD operations on a list of entities.
  - Execution of a query and presentation of the results.
  - Uploading files
- Creation of panels that show objects from meta-data.
- The transaction context of the current request is held on Bo2WicketRequestCycle which provides static methods that allow to open workers, get the thread local provider, re-attach entities etc.
- Each UI event can be mapped to a normal java method of the web layer code. This method
  - (optional) re-attaches the working objects to the persistence context
  - (optional) uses one or more business layer workers
  - performs changes to the UI

INTERAMERICAN
PART OF ACHMEA

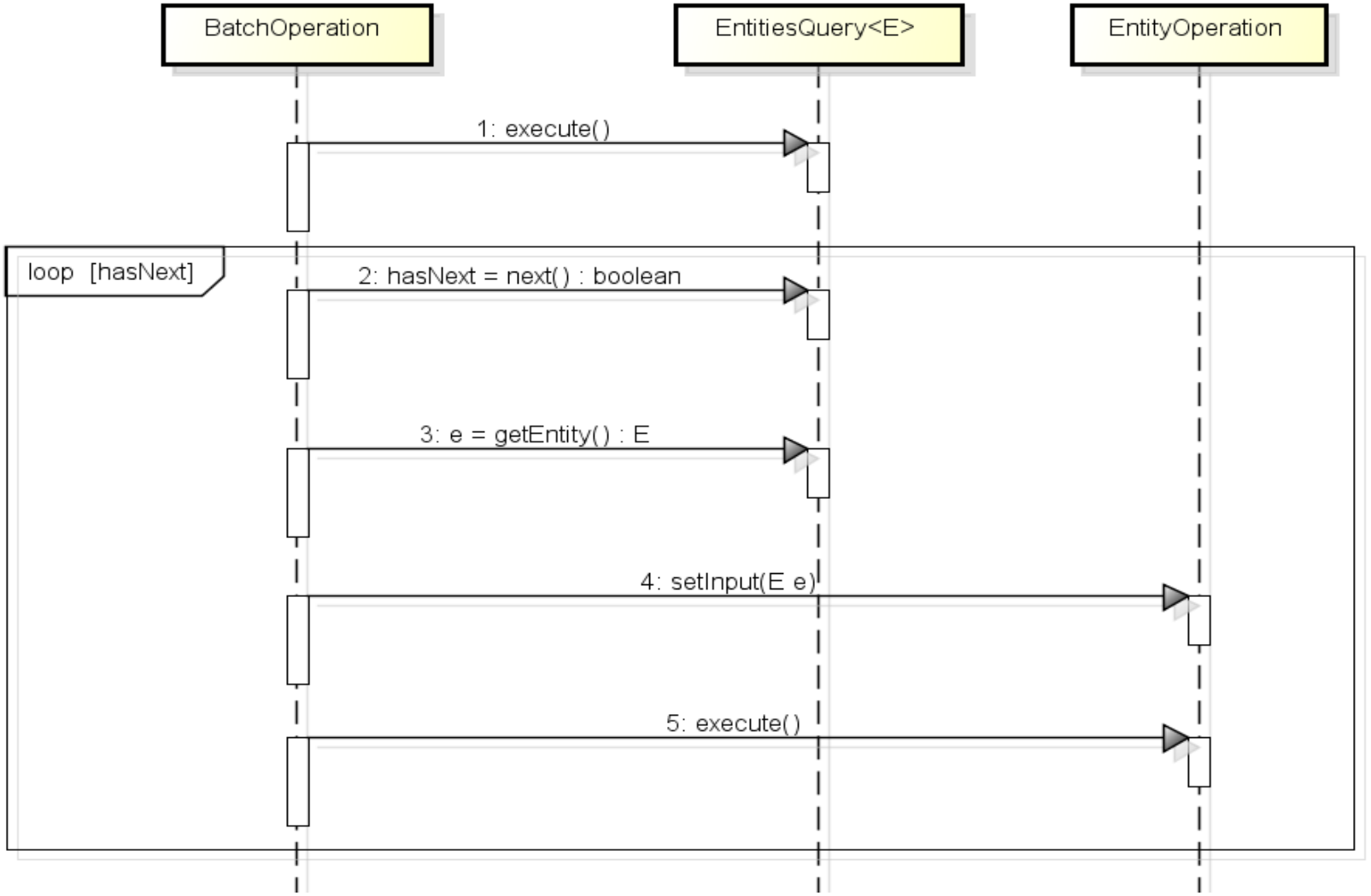
RuntimeCommand

- Used mainly on batch programs, even though the web layer equivalent adopts virtually the same idea.
- Runs a number of Operations sequentially (usually only one).
- Creates and injects the Provider for the Operations and their Worker hierarchy.
- Coordinates transaction management of transactional resources.
- May be initialized with a non-default deployment configuration. This allows, for instance to run batch operations from the container, using a scheduling framework like Quartz that provides the technology to spawn safely new Threads inside a JEE container.

INTERAMERICAN
PART OF ACHMEA

The Batch Process pattern

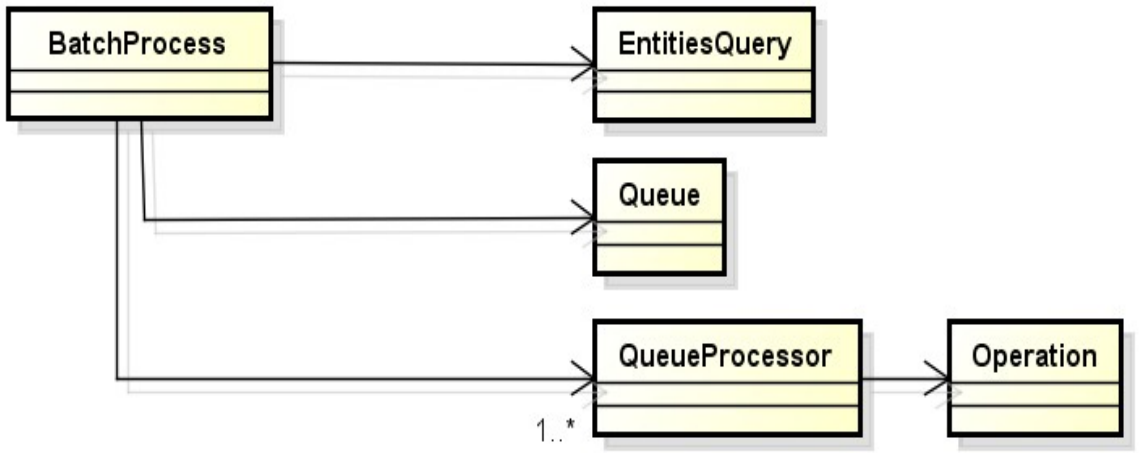
- Batch programs commonly perform the same operation on a defined set of input data.
- The input dataset can be produced by an EntitiesQuery<E> implementation.



INTERAMERICAN
PART OF ACHMEA

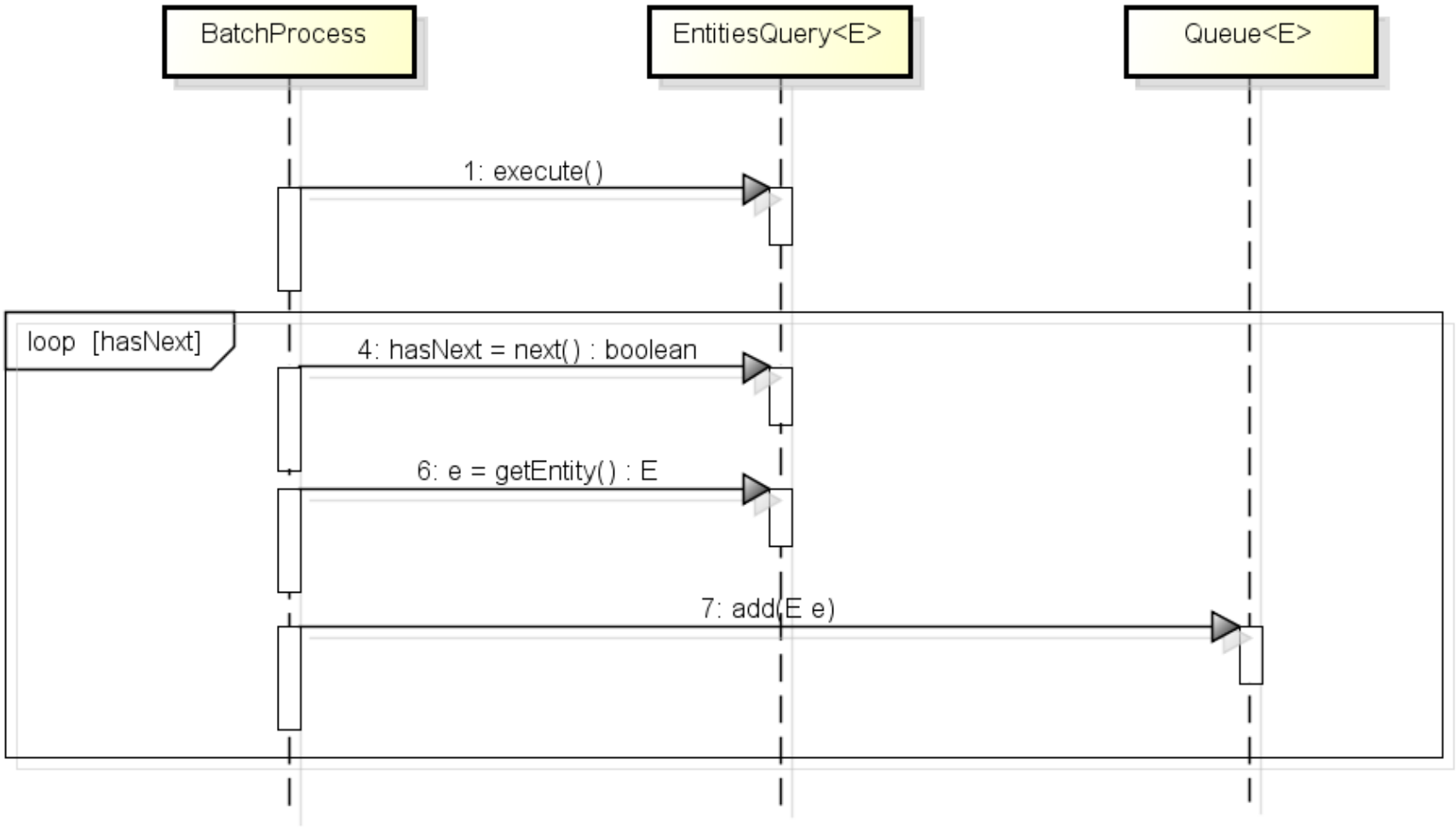
Enhanced BatchProcess

- It is not desirable to run the batch process in an 'everything or nothing' transaction. Instead, we want to demarcate separate transactions for each entity produced by the query.
- To increase throughput BatchProcess allows to perform processing in a multi-threaded environment following the Producer (the query) – Consumer pattern. The main thread runs the query and provides input to spawned threads that process it in a unit of work.
- A GUI for inspecting and administering the process is provided.



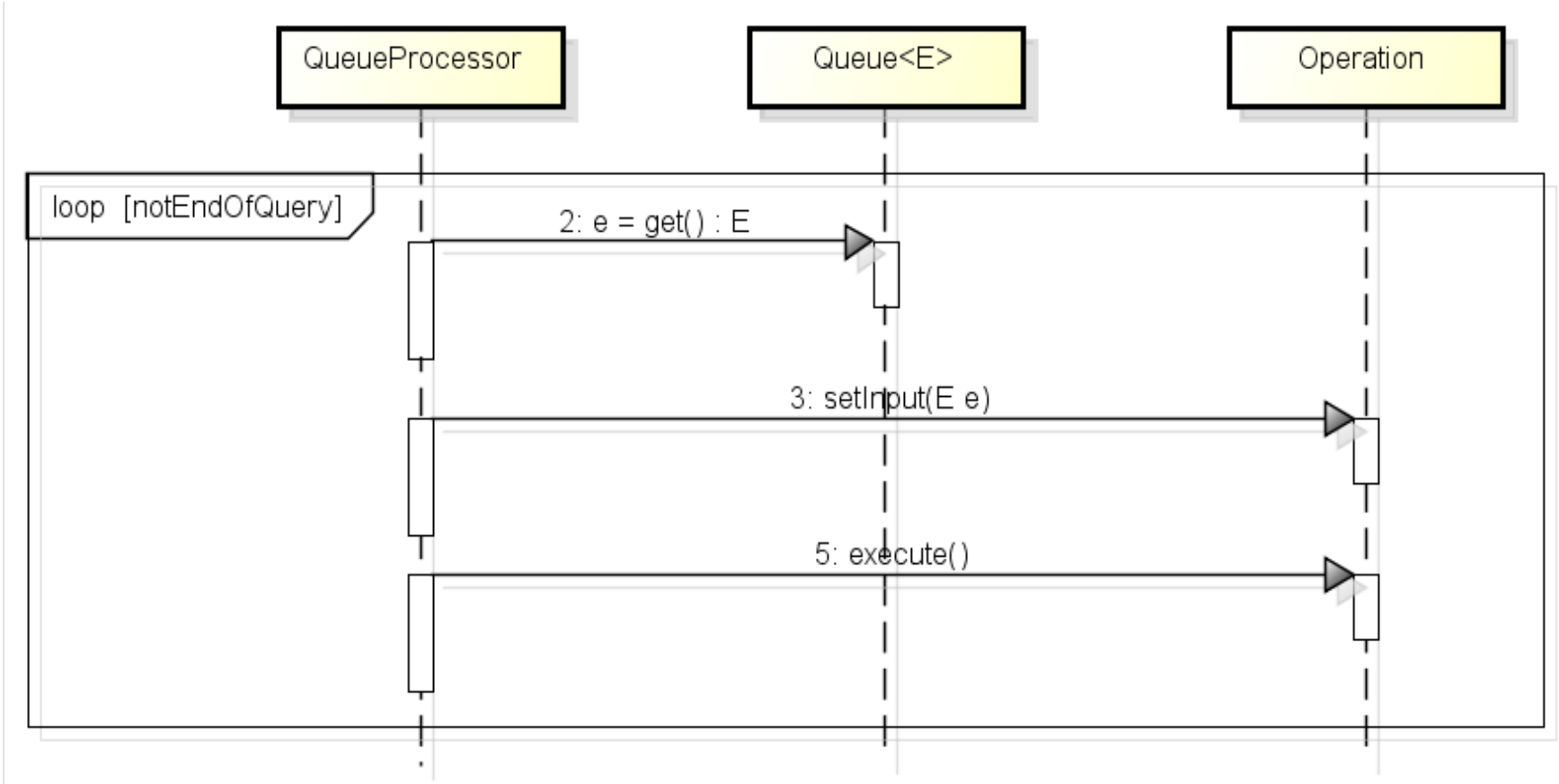
INTERAMERICAN
PART OF ACHMEA

BatchProcess: Fetched entities put to Queue



INTERAMERICAN
PART OF ACHMEA

BatchProcess: Queue elements processed



INTERAMERICAN
PART OF ACHMEA

Bo2 configuration (1)

- Introduces managers. A Worker that consumes resources is associated with a manager that provides a specific set of resources (database connection, filesystem streams etc.). A Worker using two other Workers with different managers may perform operations on multiple resources of the same type. Transaction management of multiple transactional resources is transparent to the developer.
- Definition of custom associations of interfaces with implementation classes and custom (non-hibernate) persistence workers.
- Definition of initializers. These are classes that execute code for the system wide setup upon the creation of the first Provider.
- Configuration is performed mostly with properties files. To reduce configuration duplication it is possible to inherit dynamically from base properties files and configure most of the stuff that change between deployments (e.g. database urls) in a single file.

INTERAMERICAN
PART OF ACHMEA

Bo2 configuration (2)

- Important files:
  - deployment.properties: Basic configuration file that points to the rest configuration files of the deployment.
  - pwTypes.txt: List non default persistent object interfaces → persistence worker implementations associations.
  - types.txt: List non default interfaces → implementations associations.
  - managers.txt and managerAliases.properties: Lists all available physical managers and manager names that are currently aliases.
  - of.properties: Configuration of an object factory. Each manager has an ObjectFactory instance that creates its resources. Information for the creation of resources is here.
  - vars.properties: Lists properties that are imported by other properties files so that most of the configuration that is frequently edited across deployments is in a single file.

INTERAMERICAN
PART OF ACHMEA

Q & A

Thank you
for your attention!

Any questions?