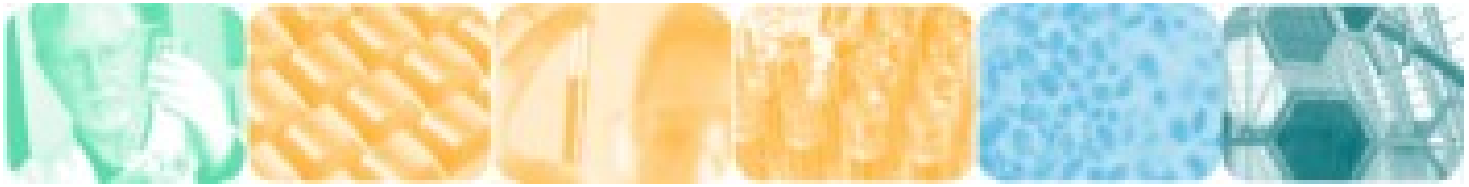
Making development of chemistry based applications easier

new (and old) languages and frameworks on the Java Virtual Machine

Tim Dudgeon <tdudgeon@chemaxon.com>



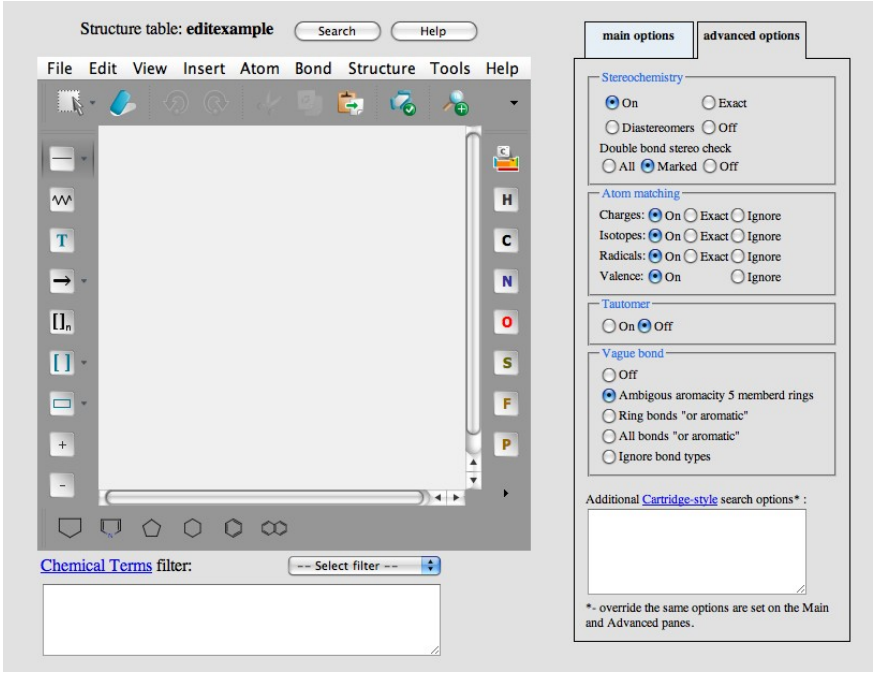
Acknowledgements

- Gustavo Santucho
- Gabriel Moreno
- Mario Burdman
- Chris Palmer

Aim

- Develop chemistry based web applications using modern frameworks
  - let the framework do the heavy lifting
  - obtain real code re-use
  - insulate ourselves from the implementation details
- Current approaches don't come close to this
- Can we improve on this?

Example 1: Structure query




```
<td>
  <select name="type" onchange="searchTypeChanged()">
    <%
    if (tableType != DatabaseProperties.TABLE_TYPE_QUERY_STRUCTURES) {
    %>
    <option value="Substructure">Substructure</option>
    <%
    }
    %>

    <%
    if (tableType != DatabaseProperties.TABLE_TYPE_MARKUSH_LIBRARIES) {
    %>
    <option value="Superstructure">Superstructure</option>
    <%
    }
    %>

    <option value="Full">Full</option>
    <option value="Full fragment">Full fragment</option>
    <%
    if (descriptor_names.length>0) {
    %>
    <option value="Similarity">Similarity</option>
    <%
    }
    %>
    <option value="Duplicate">Duplicate</option>
  </select>
</td>

...
```

```
<script language="JavaScript1.1">
<!--
   msketch_name="msketch";
   msketch_begin("../../marvin", 460, 460);
   msketch_param("molbg", "#F0F0F0");
   msketch_param("implicitH", "off");
   msketch_param("undo", "50");
   msketch_param("mol","<%=HTMLTools.convertForJavaScript(molFile)%>");
   //msketch.param("debug", 2);
   msketch_end();
   //-->
</script>
```

Example 2: JChem/JDBC DAO

Lots of code to write
Lots of traps
Very implementation specific

```
public void updateStructure(Structure structure, Connection connection) throws EngineException {
    ConnectionHandler connectionHandler = new ConnectionHandler();
    connectionHandler.setConnection(connection);
    try {
        String annotatedTableName = vendorUtils.getAnnotatedTableName(structure.getClass());
        UpdateHandler updateHandler = new UpdateHandler(
                connectionHandler, UpdateHandler.UPDATE, annotatedTableName, "");
        try {
            updateHandler.setStructure(structure.getMolecule());
            updateHandler.setID(structure.getId().intValue());
            updateHandler.execute();
        } finally {
            updateHandler.close();
        }
    } catch (Exception ex) {
        throw new EngineException(ex);
    }
}
```

Various problems

- Lots of code needs to be written
- Lots of traps for the unwary
- Much testing needed
- Difficult to test
- Difficult to refactor or upgrade
- Leads to copy & paste coding rather than code re-use
- Specialist knowledge needed
- Strongly coupled to particular database and chemistry engine implementation

History of Java web based systems

- Original JSP - Model 1 architecture. JSP demo is example. Code mess.
- Model 2 (Web MVC). Struts was the paradigm. Improved architecture, but 'configuration hell' and poor productivity
- Improved model 2 (Stripes, Spring MVC) . Made use of CoC and DRY. Improvement but still some limitations
- More OO based approaches (GWT, Wicket). More object orientated in nature.

Outside world was also doing things

- Ruby on Rails set the new standard
  - Framework designed for complete app stack
  - CoC
  - DRY
  - More productive language
- Older languages
  - Python + DANJO/TurboGears
  - PHP
- Newer frameworks on JVM
  - Groovy + Grails
  - Scala + Lift

New langauges on JVM

- New languages: Groovy, Scala, Clojure, JavaFX, JRuby, Jython
  - more fluent and productive languages (less code needs to be written)
  - dynamic, scripting and functional languages in addition to traditional Java
  - domain specific languages (DSLs)
- **JVM** not the Java language is now at the heart of the Java ecosystem
- **Groovy** and **Scala** of particular interest as they are built on top of Java - all the power of Java but with a more productive language

More is possible for less

- The framework does the heavy lifting for you
  - persistence
  - query
  - scaffolding
  - AJAX, web services and instrumentation need "turning on" rather than developing

Well designed frameworks allow

- less code to be written
- a more component based approach
- better designed systems

Too much choice can be confusing

- **XML**: DOM, SAX, TRAX, DOM4J, JDOM, Castor, JAXB ....
- **Web frameworks**: Struts, Stripes, Seam, GWT, Grails, Wicket, Tapestry, Spring MVC, JSF ...
- **Web components**: YahooUI, DOJO, jMaki ...
- **AJAX**: Prototype, jQuery, Ext...
- **Persistence**: JDBC, JDO, JPA, Hibernate, Toplink, Cayenne ...

“Standard” systems

- Most standard systems have simple data that is relatively well suited for handling in a relational database:
  - bank account, purchase orders, catalog items, addresses, flights
- Persistence tier has now been largely "automated" using ORM tools that handle CRUD and query operations largely through configuration
- You would not consider building a "standard" system nowadays without a ORM tool

Chemistry software development seems to be stuck

- Not benefiting from recent developments
  - lovingly “hand written”
  - poor productivity
- Why is this?
- Can the problems be solved?

What makes chemistry difficult?

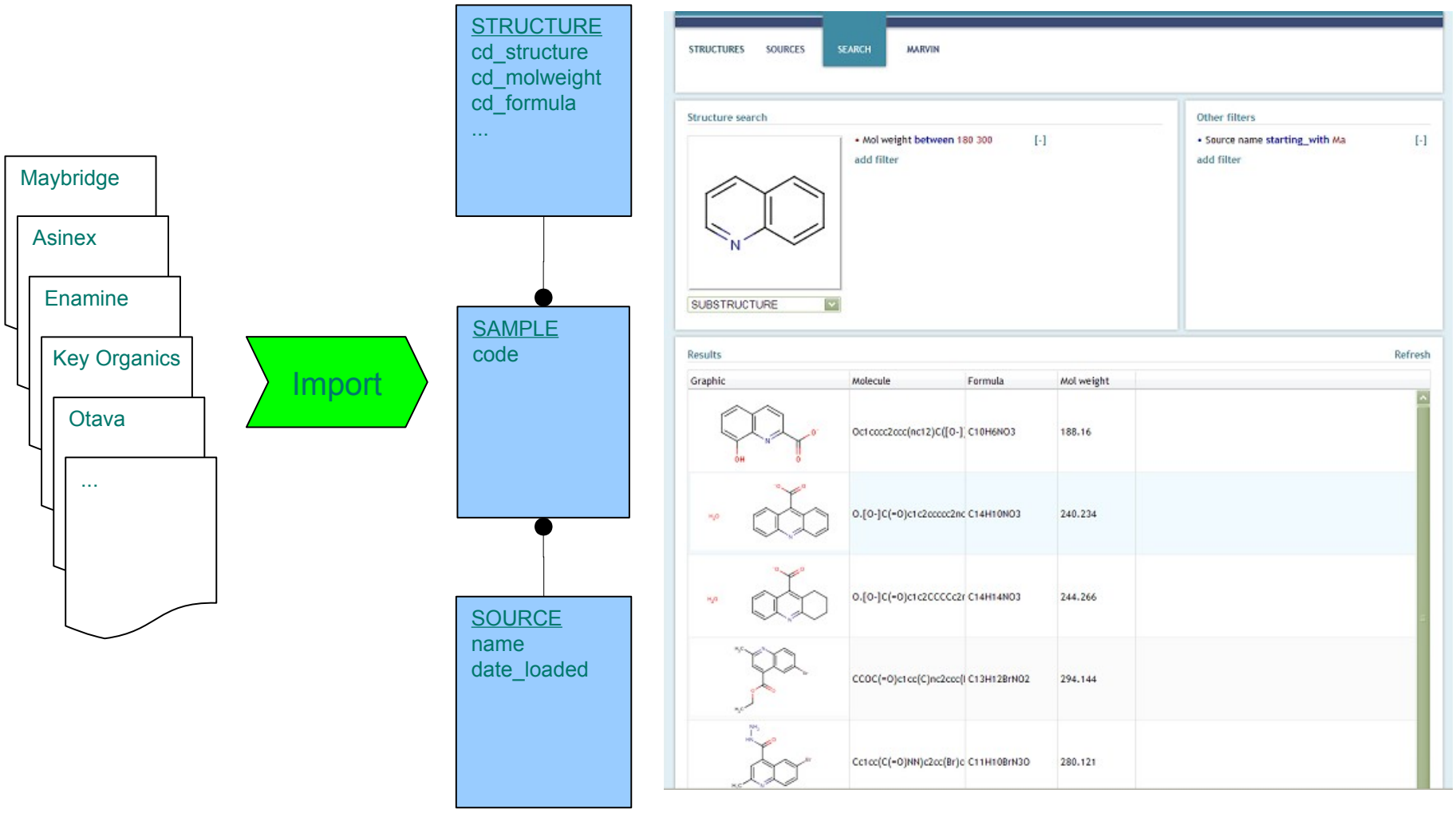
- Molecules are very complex
  - CRUD and especially query cannot be expressed as simple SQL
  - Need a “chemistry engine” to help
- As a result chemistry (Marvin/JChem) systems are typically
  - still mostly JDBC, JSP based
  - "hand written" rather than use of modern frameworks
  - much use of copy and paste code
  - make little use of components
  - need lots of code to deliver functionality
  - difficult to upgrade, refactor, test ...

Does it have to be this way? We investigate

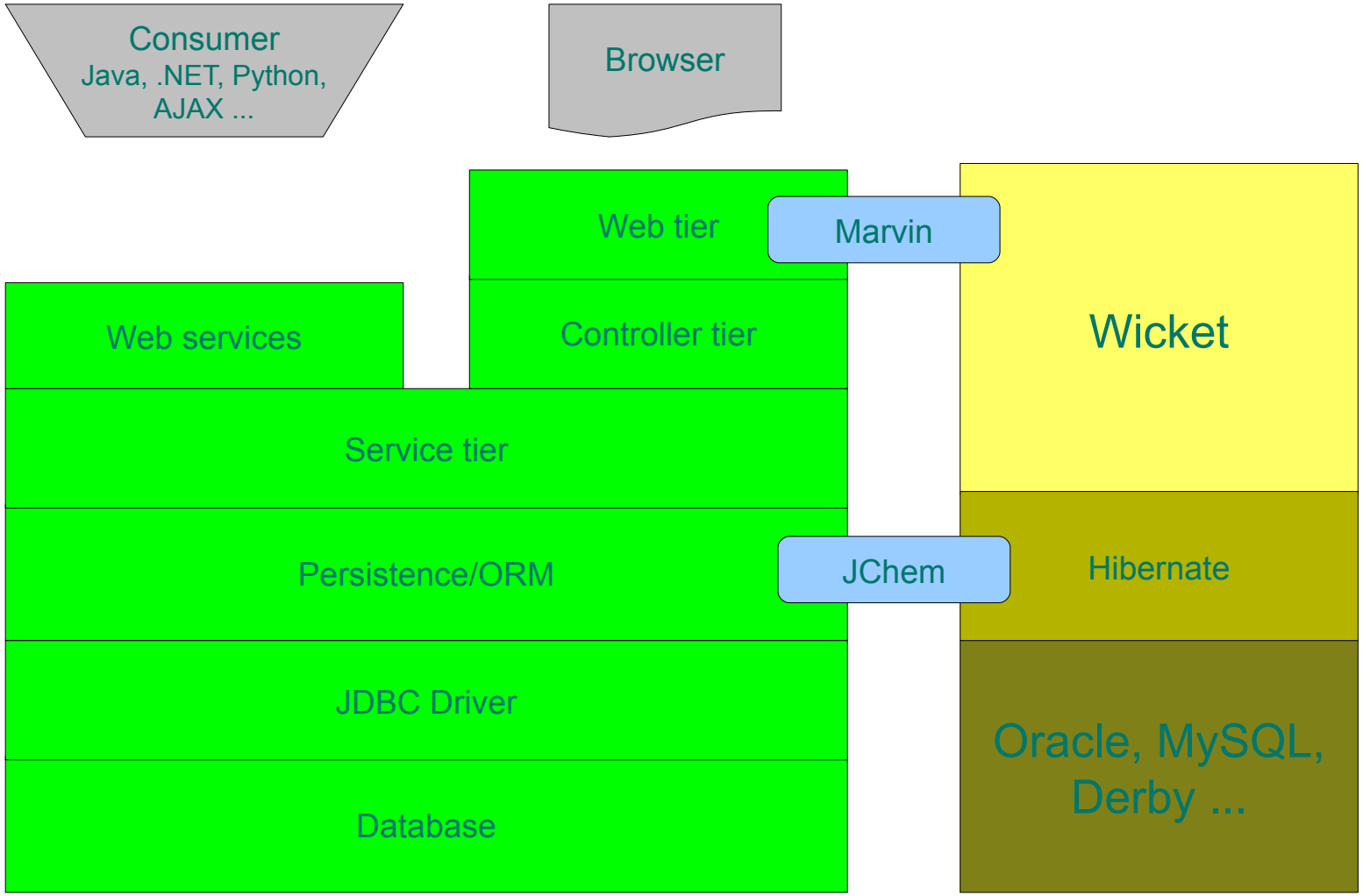
- Are there benefits in languages other than Java?
- Can persistence frameworks be used with JChem?
- Can we benefit from modern web frameworks?
- Can we get real code re-use?
- Can we get real database and chemistry "engine" independence?

=> Aim is to exemplify building a chemistry system using some of these "best of breed tools"

Example system: suppliers database



Architecture



Ideal situation: Grails Source & Sample entities

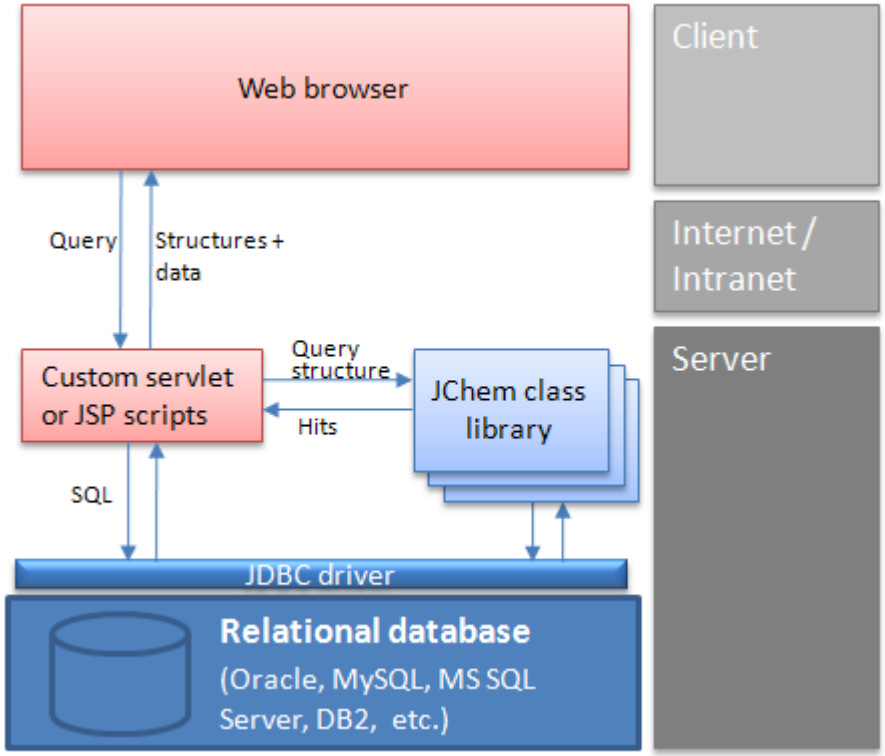
- CRUD and query operations are automatically injected
- Can this also be done for Structure?

```
class Source {

    static hasMany = [
        samples : Sample ]

    String name
    String type
    Date entered
}



class Sample {

    static belongsTo = [
        compound:Compound,
        source:Source]

    String code
}
```

```
def s = Sample.findByCode('AC 10033')
s.delete()
```

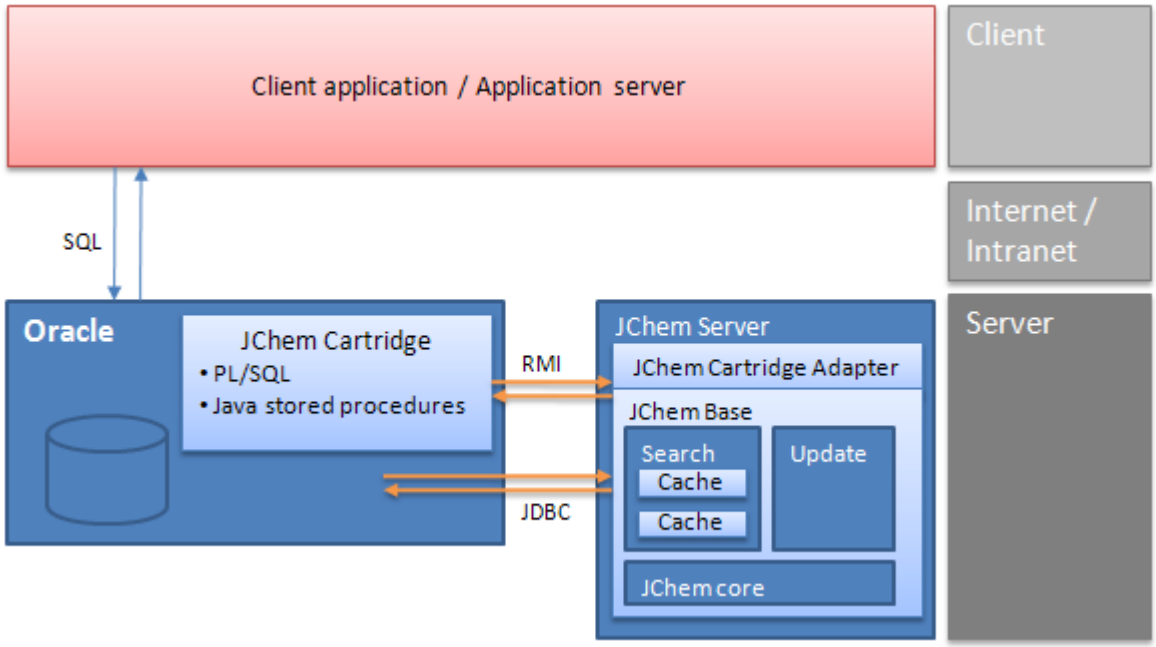
Problem with JChem base tables

- Needs use of JChem API
  - ConnectionHandler
  - UpdateHandler
  - JChemSearch
- Also uses plain SQL
- DDL operations also need JChem API
- So not obviously suited to SQL generation based persistence frameworks



JChem cartridge is a bit easier

- All operations can be done with (non-standard) SQL
  - jc_formula(), jc_compare(), jc_evaluate() ....
- SQL generator can be customised to handle this



Grails Structure entity (cartridge)

- Solves CRUD aspects
- Custom solution for query still needed
  - but named SQL queries provide a mechanism for this
- But solution only applicable to Oracle + cartridge

```
class Compound {

    static hasMany = [ samples : Sample ]

    static mapping = {
        structure type:'text'
        molformula formula:'jc_formula(structure)'
        molweight formula:'jc_molweight(structure)'
    }

    String structure
    String molformula
    Float molweight
}
```

Can we have a more generic solution?

- Can JChemBase be "tamed" and made to work with Hibernate /JPA
- Can CRUD operations on structures and non-structure entities be performed in same way?
- Can the same be done for queries?
- Can the same be done for DDL operations?
- Can the developer work with this in a "engine agnostic" manner?

Hibernate event model

Package org.hibernate.event

This package defines an event framework for Hibernate.

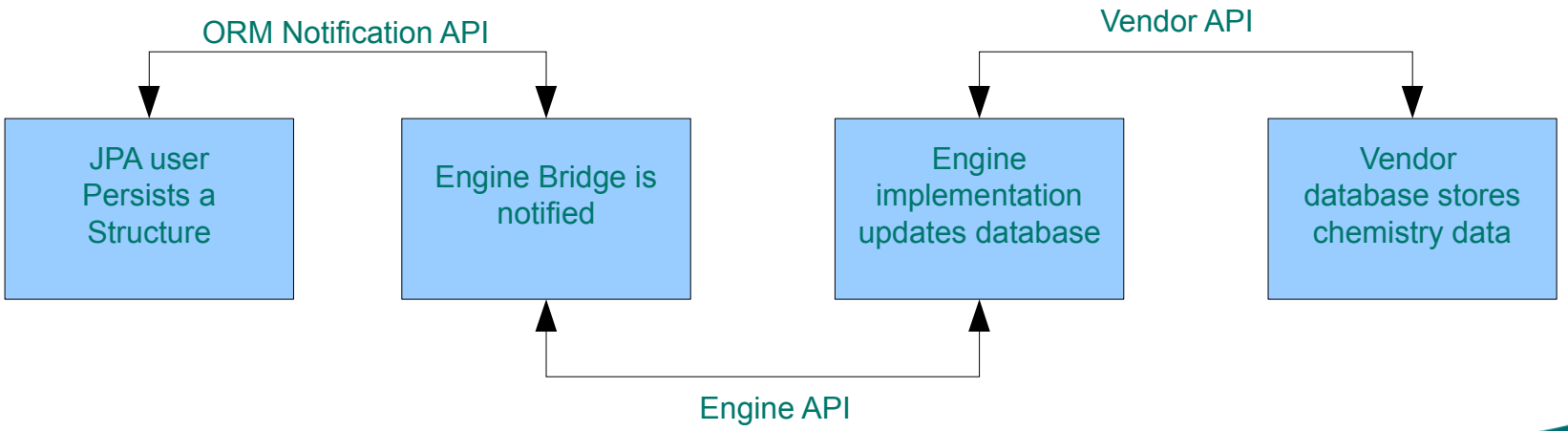
See:
Description

Interface Summary	
AutoFlushEventListener	Defines the contract for handling of session auto-flush events.
DeleteEventListener	Defines the contract for handling of deletion events generated from a session.
DirtyCheckEventListener	Defines the contract for handling of session dirty-check events.
EventSource	
EvictEventListener	Defines the contract for handling of evict events generated from a session.
FlushEntityEventListener	
FlushEventListener	Defines the contract for handling of session flush events.
Initializable	An event listener that requires access to mappings to initialize state at initialization time.
InitializeCollectionEventListener	Defines the contract for handling of collection initialization events generated by a session.
LoadEventListener	Defines the contract for handling of load events generated from a session.
LockEventListener	Defines the contract for handling of lock events generated from a session.
MergeEventListener	Defines the contract for handling of merge events generated from a session.
PersistEventListener	Defines the contract for handling of create events generated from a session.
PostDeleteEventListener	Called after deleting an item from the datastore
PostInsertEventListener	Called after insterting an item in the datastore
PostLoadEventListener	Occurs after an an entity instance is fully loaded.
PostUpdateEventListener	Called after updating the datastore
PreDeleteEventListener	Called before deleting an item from the datastore
PreInsertEventListener	Called before inserting an item in the datastore
PreLoadEventListener	Called before injecting property values into a newly loaded entity instance.
PreUpdateEventListener	Called before updating the datastore
RefreshEventListener	Defines the contract for handling of refresh events generated from a session.
ReplicateEventListener	Defines the contract for handling of replicate events generated from a session.
SaveOrUpdateEventListener	Defines the contract for handling of update events generated from a session.

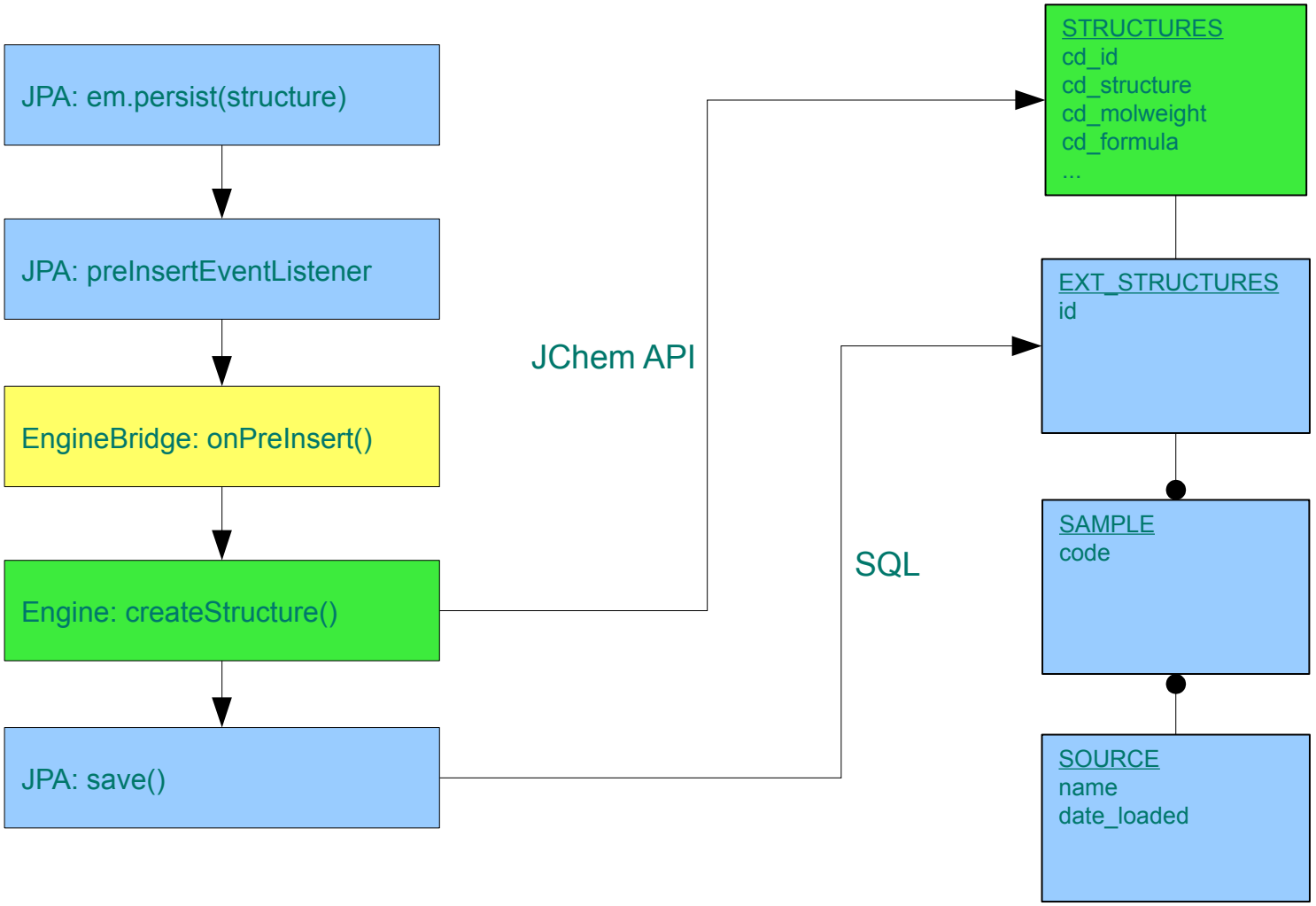
Hooks that allow persistence strategy to be customised

Chem Engine

- Engine Bridge links the JPA and chemistry worlds by implementing the ORM's notification contract and coordinating the Engine for the chemical database operations
- Engine implementation is in charge of the chemical tables through the vendor interface
- The data flows back to the user in a transparent way



Example: JChemBaseEngine



Engine bridge

```
public class EngineBridge
        implements PreInsertEventListener, PreUpdateEventListener,
        PostLoadEventListener, PostDeleteEventListener {

    private static final Logger logger = Logger.getLogger(EngineBridge.class.getName());
    private Engine engine;

    private Engine getEngine() {...}

    @Override
    public boolean onPreInsert(final PreInsertEvent event) {...}

    @Override
    public boolean onPreUpdate(PreUpdateEvent event) {...}

    @Override
    public void onPostDelete(PostDeleteEvent event) {...}

    @Override
    public void onPostLoad(PostLoadEvent event) {...}
}
```

Example implementation

AbstractJChemBaseEngine.java

```
@Override
public void createStructure(Structure structure, Connection connection) throws EngineException {
    try {
        doCreateStructure(structure, connection);
    } catch (Throwable t) {
        throw new EngineException(t);
    }
}

private void doCreateStructure(Structure structure, Connection connection) throws Exception {
    ConnectionHandler connectionHandler = new ConnectionHandler();
    try {
        connectionHandler.setConnection(connection);
        String annotatedTableName = vendorUtils.getAnnotatedTableName(structure.getClass());
        UpdateHandler updateHandler = new UpdateHandler(
                connectionHandler, UpdateHandler.INSERT_WITH_ID, annotatedTableName, "");
        try {
            updateHandler.setStructure(structure.getMolecule());
            updateHandler.setID(structure.getId().intValue());
            updateHandler.execute();
        } finally {
            updateHandler.close();
        }
    } finally {
        connectionHandler.setConnection(null);
        connectionHandler.close();
    }
}
```

Structure.java

```java
@Entity
@Table(name = "ext_structures")
@StructureEntity(tableName = "structures", tableType = TableTypeConstants.TABLE_TYPE_MOLECULES,
absoluteStereo = true, standardizerConfig = Structure.STANDARDIZER)
@XmlAccessorType(XmlAccessType.PROPERTY)
@XmlRootElement
public class Structure extends AbstractEntity {

    protected static final String STANDARDIZER =
            "<?xml version='1.0' encoding='UTF-8'?>"
            + "<StandardizerConfiguration><Actions>"
            + " <Aromatize ID='Aromatize' Type='general'/>"
            + " <Transformation ID='PlusMinus' Structure='[*+:1]-[*-:2]&gt;&gt;[*:1]=[*:2]'/>"
            + " <Transformation ID='PlusMinusDouble' Structure='[*+:1]=[*-:2]&gt;&gt;[*:1]#[*:2]'/>"
            + " <Transformation ID='azide' Structure='N=[N:1]#[N:2]&gt;&gt;N=[N+:1]=[#7-:2]'/>"
            + " <RemoveExplicitH ID='RemoveExplicitH' Groups='target'/>"
            + "</Actions></StandardizerConfiguration>";
    private String molecule, formula;
    private Float molWeight;
    private List<Sample> samples;

    @Transient
    public String getMolecule() {...}

    public void setMolecule(String molecule) {...}

    @Transient
    public String getFormula() {...}

    public void setFormula(String formula) {...}

    @Transient
    public Float getMolWeight() {...}

    public void setMolWeight(Float value) {...}

    @OneToMany(mappedBy = "structure")
    @XmlTransient
    public List<Sample> getSamples() {...}

    public void setSamples(List<Sample> samples) {...}
}
```

Using it

ChemRegService.java

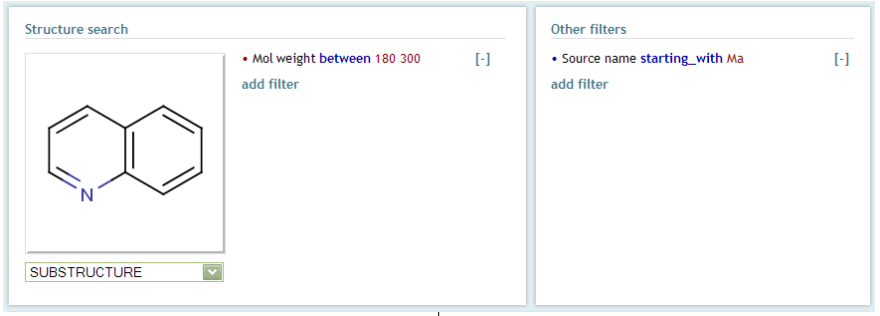
```
public Structure createStructure(Structure structure) {
    if (structure.getSamples() == null) {
        structure.setSamples(new ArrayList<Sample>());
    }
    em.persist(structure);
    return structure;
}

 public Structure findStructure(Integer structureId) {
    return em.find(Structure.class, structureId);
}
```

Plugability

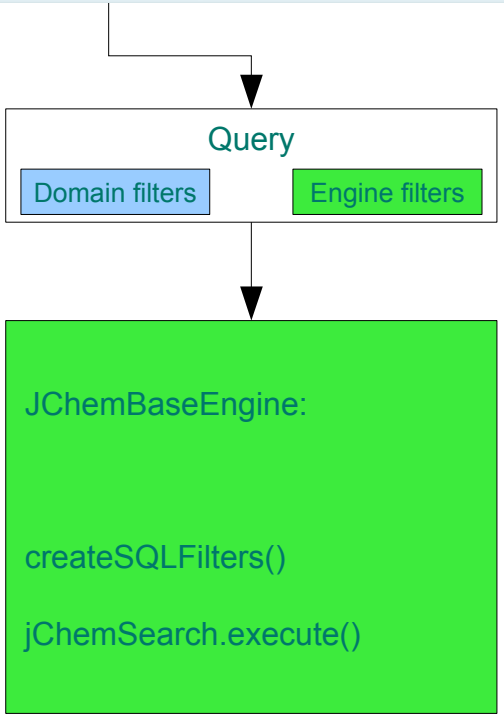
- Hibernate isolates you from the underlying DB
- Engine isolates you from the chemistry engine

```
<beans>
   <alternatives>
      <class>com.im.chemreg.chemaxon.JChemCartridgeEngine</class>
   </alternatives>
...
</beans>
```

Query (simplified)



- Non-structure queries can just use standard JPA-QL etc.
- Structure queries need special treatment
- Query inspected and translated into form that can be executed by the engine
- Result is “hit list”, not fully loaded entities (would be performance bottleneck)



Web services

1) Enable RESTful Web Services: Jersey is a Servlet

```
<servlet>
    <servlet-name>jersey</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>com.im.chemreg.harness.webservices</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>jersey</servlet-name>
    <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
```

2) Expose a functionality

```
@Path("DemoWebService")
public class DemoWebService {

    @GET
    @Path("list100Structures")
    @Produces("application/json")
    public List<Structure> list100Structures() {
        return BeanManagerLookupExtension.lookupBean(ChemRegService.class).list100Structures();
    }
}
```

3) Use it

```
def url = "http://localhost:8080/webservices/DemoWebService/list100Structures"
def WebResource webResource = Client.create().resource(url)

webResource.get(new ListOfStructures()).each {
    println "${it.id} ${it.molecule}"
}
```

Demo of suppliers system

- See it in action …
- … yes, it really works !

Did we succeed?

- Persistence
  - Approach seems successful
  - Created meaningful Web App that uses clean JPA interfaces
  - Vendor agnostic
  - Pluggable chemistry engine
- Web tier
  - Less done here
  - Limited number of Wicket components
  - Too many view technologies to choose from?

Key components that are being generated (short and medium term)

- Engine architecture and implementations
- groovy-jchem library
- Web components
  - JSP taglib, GSP, Wicket....
  - Sketcher, structure display, search options
- Other suggestions welcome

What will be available?

- These tools are currently at an early stage of development
- Will be used internally for
  - creating systems
  - consultancy operations
- Components will become available as extension to JChem (similar to web services extensions)
- Actively looking for people wanting to use