Integration with Your Projects

Mike Nowak
mnowak@umich.edu

Session agenda

- Five integration questions to answer
- Converting your data to work with the MTS Engine
- Four strategies for running the tailoring engine
- Review tailoring engine API
- Simple example: MPOWER

- Next session: lots of case studies
How do I fit this process into my project?

Five integration questions to answer

Input
1. How will I collect my data?
2. Where will my data be stored and how do I get to them?

Processing
3. What’s my strategy for running the MTS Engine?
4. What triggers the tailoring process?

Output
5. How do I handle the tailored output?
Converting your data to work with MTS Engine

- Probably the hardest part of engine integration
- Need to convert your data for each participant into a Subject object
  - data_sources: list of data sources
  - primary_chars: mapping of characteristic names to values
  - selection_chars: like primary_chars, used in selection logic
  - message_chars: like primary_chars, used in substitution process

- We provide methods for converting JSON strings, python files and Java maps into Subject objects.
- You will make your changes in SubjectFactory.py or project.py.

Another view of the tailoring process

<table>
<thead>
<tr>
<th>Deployment scenario</th>
<th>Java host application</th>
<th>HTTP web responder</th>
<th>Shell demo.py</th>
<th>Python host application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tailoring package</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming language</td>
<td>Standard library additions</td>
<td>Jython</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows</td>
<td>Mac</td>
<td>Linux</td>
<td></td>
</tr>
</tbody>
</table>

University of Michigan Tailoring Workshop
Four strategies for running the tailoring engine

- Create an HTTP interface around the MTS Engine
  http://chcr.umich.edu:51000/efl/mot/NL1/intervention/layouts/EFLmot1i-01?userid=db:XX00
  - Project Quit Phase 1 & 2
  - MENU
  - Eat for Life
  - SteppingUp
  - fMRI study
- Use the MTS Engine (tailoring2) module directly in a Python application
  - MPOWER
- Embed the MTS Engine in a Java application using Jython
  - MTS Workbench
- Create a command line interface around the MTS Engine

Review Python API: Methods and objects

- Pipeline object has methods to do selection, substitution and output step by step or in one step. See pipeline.py for details.

- select(self, message=None) returns messages with selection markup and errors
- substitute(self, selected_messages) returns selected messages with substitution and errors
- render(self, selected_messages) returns HTML and errors
- render_from_scratch(self) returns does all three steps and returns HTML and errors.
Review Python API: Sample code

```python
# See tailoring2/scripts/demo.py for an example
import tailoring2

## load subject and compute derived
subject = project.subject_for_primary_chars(primary_chars)
pipeline = new Pipeline(message, subject, data_source)

## selection
selected_messages, errors = pipeline.select()

## substitution
subbed_messages, errors = pipeline.substitute(selected_messages)

## rendering
result, errors = pipeline.render(subbed_messages)

## or all at once
result, errors = pipeline.render_from_scratch()
```

SubjectFactoryType is a Java wrapper for a python SubjectFactory. Create with SubjectFactoryFactory.create()

SubjectData is a java object that contains a list of sources and their values

SubjectResult is an object generated by the engine, contains a Subject object and a List of errors. Create with SubjectFactoryType.subjectForSubjectData(subjectData)

Subject is a java object containing data sources, primary selection and message characteristics. Get a Subject with SubjectResult.getSubject()
Java API: Engine objects

- **EngineType** is a Java wrapper for a python `Engine` object. Use `EngineFactory.create()` to create an `EngineType` object.

- **RenderResult** is a Java object containing the section name, rendered message in HTML, and a List of errors. Use `EngineType.renderSectionHTML()` to get a `RenderResult` object.

```java
// See PublishWizardJob.java for an example
File messageDoc, testCase;

// Get an instance of an engine and subjectFactory running in Jython
EngineType engine = new EngineFactory().create();
SubjectFactoryType subjectFactory = new SubjectFactoryFactory().create();

// create subject, using test case convenience method here
SubjectResult subjectResult = subjectFactory.subjectForTestCaseFile( testCase).getSubject();
Subject subject = subjectResult.getSubject();

// and render tailored message
RenderResult result = engine.renderDocumentHTML( messageDoc.getPath(), section, subject, source );
```
Goal: Send tailored text messages on a schedule to patients enrolled in a study at a pediatric obesity clinic.

<table>
<thead>
<tr>
<th>Data collection</th>
<th>Data storage</th>
<th>Engine integration</th>
<th>Tailoring trigger</th>
<th>Tailored output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web survey, collected in the clinic.</td>
<td>Data stored in MySQL database on web server, retrieved on demand by MPowerServer.</td>
<td>Engine runs native as part of python-based MPowerServer.</td>
<td>MPowerServer generates tailored messages daily and stores the result in the database.</td>
<td>Delivered on schedule as SMS or MMS message by Nokia N80 phone running python client.</td>
</tr>
</tbody>
</table>
class DBSubjectStrategy(object):
    """get subject from the database"""

    _old_sql_one_user = ""
    SELECT baseline.*, eligibility.*, arm.*, participant.FirstName, participant.quitdate
    FROM participant, baseline, eligibility, arm
    WHERE participant.userid = %s AND
    baseline.userid = participant.userid AND
    eligibility.userid = participant.userid AND
    arm.arm = participant.arm;
    ""

    sql_one_user = """"SELECT * FROM fmriTailoring_View WHERE ActualID = %s"""

    def get_raw_chars(self, userid):
        """query the database for the user's survey answers and personal/contact info -- return a dictionary."
        log.info(">>>dbstrategy.getrawchars(%s)" % repr(userid))
        conn = self.project.db_connection()

        # print query using MySQLdb quoting rules
        log.debug(self.sql_one_user % conn.literal(userid))

        cursor = conn.cursor()
        cursor.execute(self.sql_one_user, (userid,))
        result = cursor.fetchone()
        if not result:
            raise errors.SubjectLookupError("user 'Ks' not found in database" % userid)

        description = dtuple.TupleDescriptor(cursor.description)
        chars = dtuple.DatabaseTuple(description, result).asMapping()
        return chars
def _getsubject(self, key):
    """do the work of constructing the subject""
    raw = self.get_raw_chars(key)
    primary = apply_phase(self.coalesce_phase, raw)

    # do empty every time in case empty changes (or make underlay accept a callable)
    empty_underlay_phase = UnderlayVariablesPhase(self.project.getemptysubject().primary_chars)
    intermediate_phases = [
        empty_underlay_phase,
        self.convert_not_answered_phase,
        self.insert_not_answered_instance_phase,
        # self.personality_phase,
        # self.override_quit_date_phase,
        self.derive_phase,
    ]
    selection_chars = apply_phases(intermediate_phases, primary)
    message_chars = apply_phase(self.text_sub_phase, selection_chars)

    return Subject(selection_chars, message_chars, id=key,
                   primary_chars=primary, derived_names=self.get_derived_names(),
                   library_names=self.derive_phase.deriver.glb)