Python bindings for HLA (pyHLA)

This document does not describe any existing or planned system. It's intended for public discussion about an idea to develop Python bindings for HLA.

Goals

To integrate HLA into the Python language.
To enable rapid and easy development of HLA federates. Reduce the development and maintenance effort (compared to C/C++).
To facilitate integration of non-HLA compliant applications to HLA.

Simplify the activity 4.3 of FEDEP [IEEE 1516.3].

Motivation

(1) Limitations of the C++ API for HLA.
The HLA standard defines C++ mappings [IEEE 1516.1], but a significant development effort is necessary to develop a HLA compliant federate in C++.
The development effort can be reduced by using

- higher-level interfaces, e.g. the Protocol Independent Interface in MÄK VR-Link
- code generators, e.g. the GENESIS developed by ONERA [GENESIS]

The HLA standard covers IEEE 1516.1 only. The C++ API for value encoding [IEEE1516.2] is not standardized. Every HLA developer needs to implement the value encoding functions.

(2) Benefits of the Python language.
Dynamic data types. High-level programming language.
Interpreted. Easy modifications.
Powerful plug-in system.
Many plug-ins providing scientific calculations, geodetic conversions, etc.

Requirements

High-level. Shall encompass the most frequently used HLA features. May not encompass all of the features.

Direct integration of FOM/SOM classes and data types.
Direct import of OMT DIF files. No pre-processing.
Automatic IEEE1516.2 value encoding.

Automatic entity publication and discovery.

Example:
Connect to the RTIG.

```
connection = hla.Connection("federation", "federate", "model.fed")
```

Import data types from RPR-FOM.

```
hla.use("rpr-fom.xml")
```

All the RPR-FOM classes and data types are now directly accessible. Create the Aircraft class and set its world coordinates.

```
aircraft = hla.Aircraft()
aircraft.WorldLocation = (0,0,10)
```

Publish the aircraft.

```
connection.publishedEntities["OS-001"] = aircraft
```

List world coordinates of all discovered aircrafts. By default, all discovered entities are automatically stored in the discoveredEntities dictionary.

```
for entity in connection.discoveredEntities:
  if type(entity) == hla.Aircraft:
    print entity.WorldLocation
```

Dynamic data types. Duck-typed.
Notifications received as method calls in sub-classed functions.

Example:
Define a custom Aircraft class.

```
class MyAircraft(Aircraft):
  pass
```

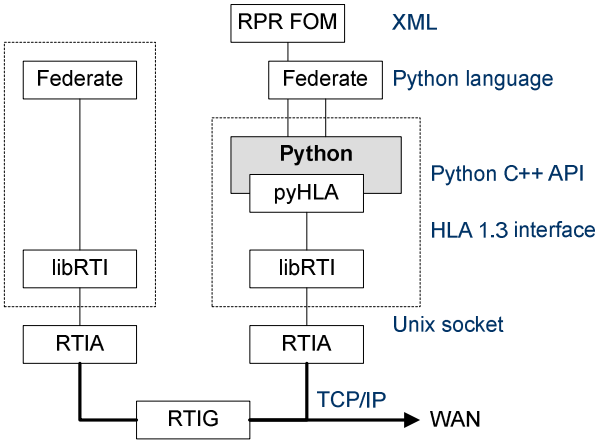
Define a custom connection that will discover aircrafts only. Each aircraft will be represented by the custom Aircraft class.

```
class MyConnection(Connection):
  def discoverObjectInstance(self, name, class):
    if class == hla.Aircraft:
      self.discoveredEntities[name] = MyAircraft()
```

Do not hide the main loop from the user. Keep the tick() function in the Python script.

Implementation

Use the Python/C API.
Use Python meta-classes to dynamically generate the FOM data types.



Performance

Discuss the performance penalty. Use the DFSS/DMAIC methodology.
Design experiments for performance measurements.
Measure performance of Python and plain C++ federates. Use multiple RTI: CERTI and MÄK RTI.
Discuss the results.