USE OF PYTHON AS A SATELLITE OPERATIONS AND TESTING AUTOMATION LANGUAGE

Gonzalo Garcia
VP of Operations, USA
INTRODUCTION
INTRODUCTION

Presentation shares the results of GMV’s experience regarding:

- **Next generation automation layer** for Satellite Command and Control (SCC)
  - Required to interact with the core of the SCC via an existing **Application Programming Interface (API)**, exporting its services as functions available from a dynamic language.
  - Layer was expected to support
    - Automation of **operational** satellite control procedures
    - Automation of **non-regression SCC testing** during development, integration and maintenance.
    - All previous points also applied to **ground equipment**

**Broad view of automation, applied well beyond actual operational procedures**

- In particular, this presentation summarizes our analysis regarding the feasibility of the use of **Python** as the scripting language.
SCC AUTOMATION
LANGUAGES
EXISTING LANGUAGES FOR SCC AUTOMATION

- SCC automation approaches:
  - **procedural scripts**
    - Space-specific languages
    - General purpose languages
  - rule-based expert systems
  - finite state models

- Multiple **space-specific languages currently used**:
  - **STOL**: Satellite Test and Operations Language
    - Originally developed by NASA, multiple flavors
    - Widely used by many GOTS and COTS
  - **PLUTO**: some ESA missions (SCOS-2000)
  - Multiple **proprietary languages** used by different companies: SOL (GMV), CCL (Harris), OCIL / CECIL (Raytheon), PIL (Astrium), SCL (ICS), etc

- **General purpose languages** used in some missions: Perl, Tcl
### CUSTOM vs GENERAL PURPOSE LANGUAGES

<table>
<thead>
<tr>
<th>SPACE-SPECIFIC (eg. STOL)</th>
<th>GENERAL PURPOSE (eg. Python)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROS</strong></td>
<td></td>
</tr>
<tr>
<td>□ (Sometimes) more user friendly for non-programmers</td>
<td>□ Open source</td>
</tr>
<tr>
<td>□ Adapted to satellite operations</td>
<td>□ Very powerful</td>
</tr>
<tr>
<td>□ High reliability</td>
<td>□ Portable</td>
</tr>
<tr>
<td>□ Language can be easily restricted / extended</td>
<td>□ Limited, enhancements are expensive</td>
</tr>
<tr>
<td>□ Proprietary language and/or tools</td>
<td></td>
</tr>
<tr>
<td>□ Portability issues</td>
<td>□ Potentially less readable if coding is not done carefully</td>
</tr>
<tr>
<td>□ Limited, enhancements are expensive</td>
<td>□ Too powerful?</td>
</tr>
<tr>
<td><strong>CONS</strong></td>
<td></td>
</tr>
<tr>
<td>□ Proprietary language and/or tools</td>
<td></td>
</tr>
<tr>
<td>□ Portability issues</td>
<td></td>
</tr>
<tr>
<td>□ Limited, enhancements are expensive</td>
<td></td>
</tr>
</tbody>
</table>
HOW ABOUT PYTHON?
Python

- Python is a **portable, open, high-level, object-oriented, dynamic language**
- Conceived in the 80s, **used massively since the 90s**
- Recognized widely for its **readability, maintainability** and **modifiability**, key aspects for complex procedures that may be modified multiple times throughout a mission.
- **Performance** is much better than most other dynamic languages.
  - Compiled to bytecode
- **Widely supported by the software community**, which guarantees the availability of good programmers, Integrated Development Environments (IDEs) and extensions.

```python
def add(x):
    return x+1

def dotest(x):
    node_name = get(node_name)
    label = symbol_sym_name.get_int(node_name, label)
    print "&\langle%" % node_name, label,
    if isinstance(node_name, str):
        if node_name is not None:
            print "%s = %s" % (node_name, label)
        else:
            print "="
    else:
        print ":=

    print ":="
    children = []
    for child_name in enumerate(node_name):
        children.append(dotest(child))
    print "%: = %" % node_name,
    for name in children:
        print "%s = %s" % name,
```

Multiple successful applications in space business
E.g. Shuttle Mission Design
ADVANTAGES OF THE USE OF PYTHON (1)

- **Portable**
  - Windows (XP, CE, Pocket PC), Linux, UNIX, Macintosh
  - Many others: AIX, AROS, AS/400, iPOD, OS/2, Palm OS, Playstation, Psion, VxWorks, Nokia cell phones, .NET, Java Virtual Machine, ...

- **Dynamic**
  - Dynamically typed and interpreted, ideal for fast scripting

- **Open**
  - Free, even for commercial use.
  - Interpreter can be embedded in products (no license fee)
  - Open source, no GPL-like traps

- **Powerful**
  - Complex built-in data structures (e.g. flexible arrays, lists, dictionaries)
  - Great variety of program control instructions
  - **Productivity 5 – 10 times higher than Java**
  - Supports exception handling
  - Automatic memory management and garbage collection
  - Language is extensible
ADVANTAGES OF THE USE OF PYTHON (2)

- **Object orientation**, with all the associated benefits (reuse, abstraction, scalability, ...)
  - Supports classes, inheritance, templates

- **Built-in development capabilities**, given as language modules
  - Automatic documentation generation
  - Unit testing, regression testing
  - Debugger, profilers, interpreter, compiler

- **Easy integration with**
  - existing Service Oriented Architecture (**SOA**) implementations
  - **Web Services** (**WSDL**)  
  - **GMSEC API** (Python supported)

- **Availability of multiple modules** for
  - **XML** processing: Multiple applications: XTCE DB parsing, SOAP messaging, etc
  - **Communications**: Sockets, Internet access, RPC, email
  - Time **performance measurement**
  - **Many others**: database access, math, data compression, multi-threading, cryptography, operating system access, etc
ADVANTAGES OF THE USE OF PYTHON (3)

- Availability of bindings for multiple **GUI-development toolkits** (Qt4, GTK2, Tk, wxWidgets, etc)

- Wide variety of **plug-ins for Eclipse** (a popular open development platform) can be used to work with Python.

- Availability of multiple, powerful, **free tools** for
  - Development
  - Source code inspection and metrics generation
  - Debugging, testing
  - Configuration management

- Wide support by **commercial tool vendors**
RISKS OF THE USE OF PYTHON

- **Language may be too powerful** and complex for non-programmers.
  - This can be handled by restricting the use of certain instructions from the development environment.

- **Readability** may be worse than space-specific languages if coding is not done carefully.
  - Strict coding standards are needed.
  - Coding can be abstracted for non-programmers using a visual environment.

- **Evolution of language** is controlled by others.
  - This is part of the deal of using a general-purpose language.
  - Compensated by all the advantages.
  - A mission can just freeze the Python version & development environment and use updates on a case-by-case basis.

- **Dependency** on third-party software (the interpreter).
  - But it is open source.
INTEGRATION OF PYTHON WITH AN SCC
INTEGRATION OF PYTHON WITH AN SCC

- A tool was created to develop, test, modify and schedule the Python procedures.
  - Target users: Satellite operators
  - Environment fully customized to take into account the target automation requirements
    - Operational procedures
    - SCC non-regression testing
    - Ground equipment operations & testing

- Access to the SCC API is enabled by a Python library that encapsulates all the standard API services
Development and Execution Environment (1)

Development environment

- **Objective**: Deliver the most powerful support for procedure development and validation
- Based on **Eclipse/RCP** (Rich Client Platform):
  - Open Development Platform
  - Widely adopted as Integrated Development Environment
  - Open source
  - Supports scripting languages
  - RCP: specifically designed to build custom IDEs

- **Capabilities**
  - Repository
  - Edition: Including syntax highlighting
  - Verification: Including procedure verification against the satellite database
  - Automatic look-up of class methods, function arguments, etc
  - Metric generation, coverage statistics
  - Debugging
DEVELOPMENT AND EXECUTION ENVIRONMENT (2)

```
Python code:

```
DEVELOPMENT AND EXECUTION ENVIRONMENT (3)

- **Procedure execution** services
  - Procedure execution (cold/warm start, start at, etc)
    - Parallel execution supported
  - Procedure control (pause, resume, step, etc)
    - Supports step-by-step execution as well as spacecraft protocol details (eg. TC verification)
  - Procedure monitoring (execution status, etc)

- **Repository browser**
  - browse (read only) validated procedures

- **Scheduler**
  - Schedule, control and monitor procedures
    - Triggers, events, pause, resume, etc
PROCEDURES (1)

- Nominally, **native procedures** written in Python
- Support for existing space-specific languages provided by the development of **conversion tools** that generate the extended Python scripts from the legacy operational procedures.
  - STOL, CECIL
  - Procedures in XML
- This is very important to **minimize cost and risk** when
  - adapting standard platform-specific procedures from certain manufacturers
  - replacing an operational SCC that used procedures in these languages
The flexibility of Python has made it possible to perform this **conversion in two ways**:

- As a **batch process**, where a set of scripts have been converted to the extended Python

- As a **real-time conversion**, allowing the operator to keep using the original language for step-by-step execution monitoring and for the implementation of modifications

Maintains **traceability** between lines of code of original procedures and translated procedures
**Python + SCC LIBRARY**

- **Python**: Includes
  - Interpreter
  - Multiple extensions supporting arrays, vectors, XML, GUIs, HTML, etc

- **SCC Library**
  - Provides access to **API services** from the Python code
  - Services are **encapsulated** in a class, services become class methods
  - Potentially **this class could be standardized** to allow the operator to:
    - Use the same procedures with different SCCs (e.g. heterogeneous fleet using different products)
    - Migration from a legacy SCC to a new product
SCC API

API Services
- Access to satellite database definitions
- **Telemetry** (TM)
  - TM access (real-time, retrieval, packets and parameters, single parameter, parameter sets)
  - TM injection
- **Telecommands** (TC)
  - TC injection (real-time, retrieval, filtering criteria)
  - TC history access
  - TC status monitoring
- **Event and out-of-limits** access (real-time, retrieval, filtering criteria)
- **Event injection**
- **Modification of out-of-limit definitions**
- **Open** predefined TM displays
LESSONS LEARNED (1)

- Many **space-specific languages** currently used for the development of operational procedures were **defined decades ago** and are **not used outside of the space industry**. In many cases they are **proprietary** and require **expensive** products.

- Future **support for proprietary languages and availability of tools** is not guaranteed. Some operators have had serious problems replacing a system once the HW became obsolete, typical in a GEO mission (> 15 years)

- Lessons from **Ada**:
  - Language designed under contract to the US DoD during 1977 – 1983
  - Targeted at embedded and real-time systems
  - Mandatory for new software DoD projects since 1987
  - Excellent language, used successfully for thousands of projects
  - 2003, Software Engineering Institute:
    "**Due to a dearth of tools and compilers and lack of trained, experienced programmers [...] Ada is a programming language with a dubious or nonexistent future**"
LESSONS LEARNED (2)

- Operators with a **heterogeneous fleet** usually end up having to use different languages. This increases training & operations costs and increases the complexity of the system.

- Python allows the definition of a **homogeneous front-end** for a heterogeneous fleet.

- **Coding rules, customized development environment and training** needed to guarantee the high quality & maintainability of procedures.

- With Python, operators can **benefit** enormously from the **software community**:
  - Using **modern, powerful, open source languages** like Python and **tools** like Eclipse/RCP widely supported.
  - Approach allows the operators to have an **open, integrated environment** for operational and testing procedure development, verification, execution, configuration management and metric generation.
  - It also **reduces** the **dependency** on proprietary technologies and the **risks** of software obsolescence.
Thank you