Karlsruhe Institute of Technology

Projektpraktikum Robotik und Automation I

Sommersemester 2018

Department of Informatics – Institute for Anthropomatics and Robotics - Intelligent Process Control and Robotics (IAR-IPR), Prof. Dr.-Ing. habil. B. Hein

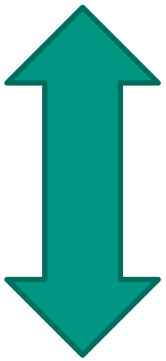
ROS - OPC UA Connector 3.0

Peiren Yang

Advisor: M. Sc. Denis Štogl

Motivation

- There Is No Industry 4.0 without OPC UA*
- ROS is a powerful robotic middle-ware and has been widely used in the robotics community
- As a new trend, ROS-Industrial allows the use of ROS in industrial automation
- Therefore it is meaningful to model ROS with OPC UA, so as to facilitate a unified access interface of ROS in industry 4.0 world



* https://www.automation.com/automation-news/article/there-is-no-industry-40-without-opc-ua

Review of ROS - OPC UA Connector 1.0

- Implemented functionalities
  - Dynamic modeling of ROS Services, Topics as well as Actions. The UA model supports:
    - ROS Service call
    - ROS Topic publish and display published content
    - Sending and canceling ROS Action, display feedback, result, status
  - Modeling of ROS Msgs, Srvs.
  - Import and export modeled UA nodes in XML format.

- Bottlenecks & Flaws
  - The framework “python-opcua “ does not support customized structures, therefore the modeled ROS Msgs, Srvs cannot be used directly.
  - Each time if a new ROS topic object is received, it should be unboxed to simple UA data types, and if a ROS service with data object should be called, the data with simple UA types has to be boxed into an object, this lead to a considerable performance loss.

New design idea

- Modeling ROS dynamic and static information separately
  - Static information: ROS Msgs and Srvs
    - They are predefined before the roscore runs
    - Used for information exchange within ROS
  - Dynamic information: ROS services, topics, actions, parameters, nodes.
    - Generated after roscore started, managed by roscore.
    - Can be created and deleted.
    - The behavior of them is unknown before one is created.
    - To Interact with them, predefined Msgs and Srvs must be used.

Example Msg:

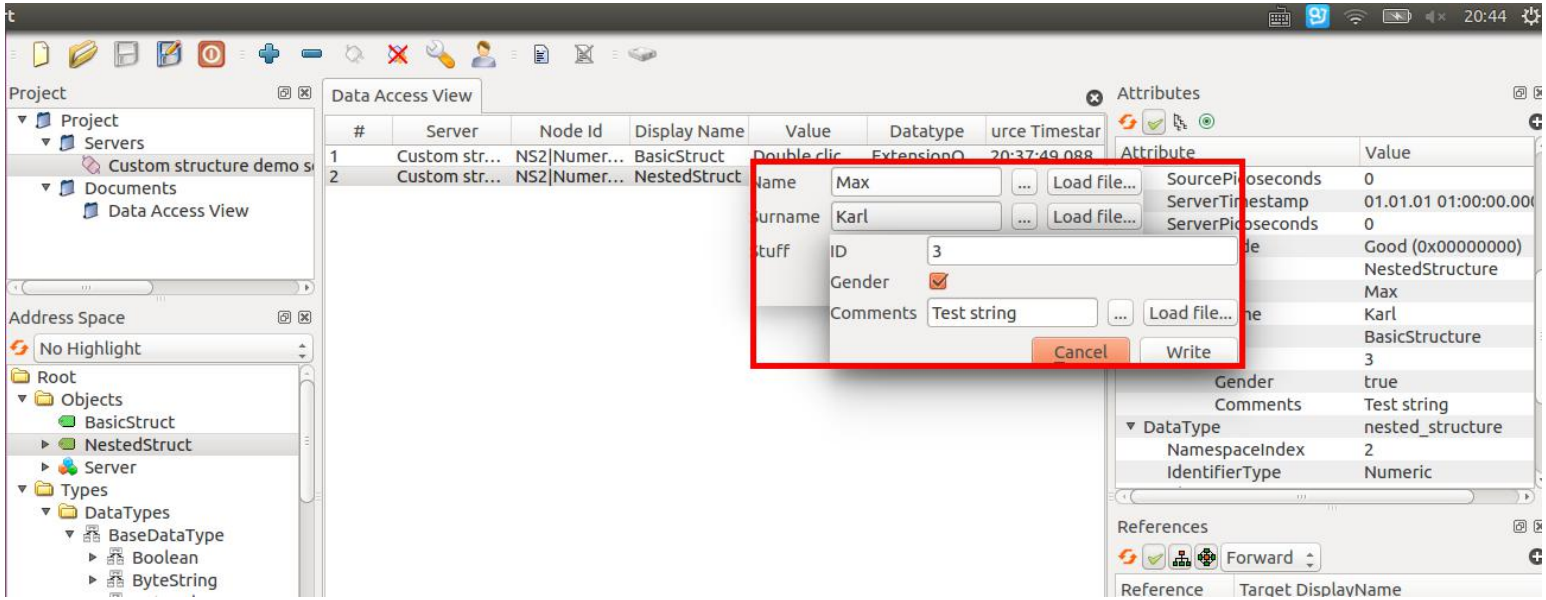
```
string first_name
string last_name
uint8 age
uint32 score
```

Example Srv:

```
int64 A
int64 B
---
int64 Sum
```

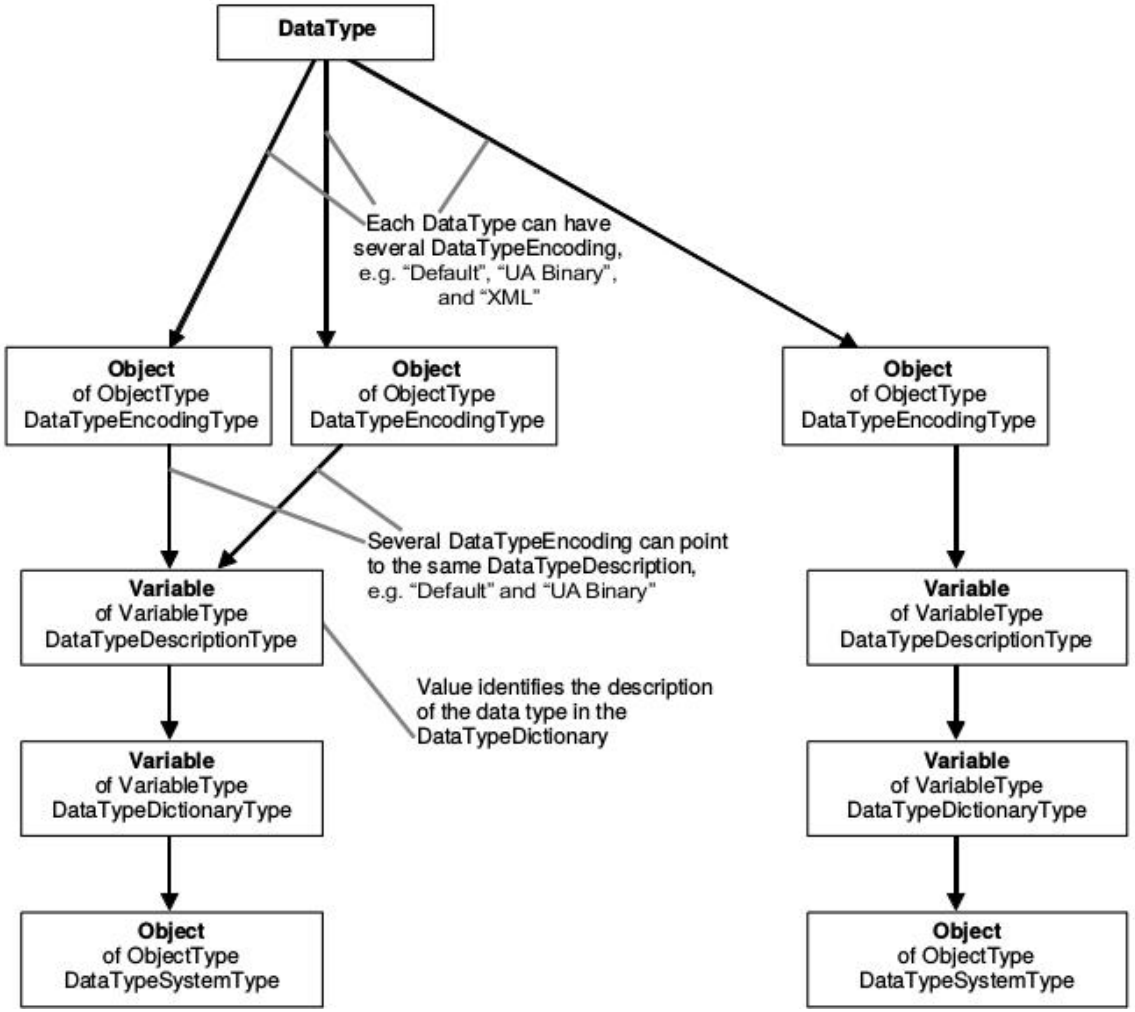
New design idea

- Extend “python-opcua”, let it support customized structures
  - Modeling all ROS Msgs and Srvs with customized structures, so that the recursive boxing and unboxing operations during dynamic running are avoided.
  - Generate python class for the newly created OPC UA structures, to make the data transfer in UA world possible.



The OPC Type Description System*

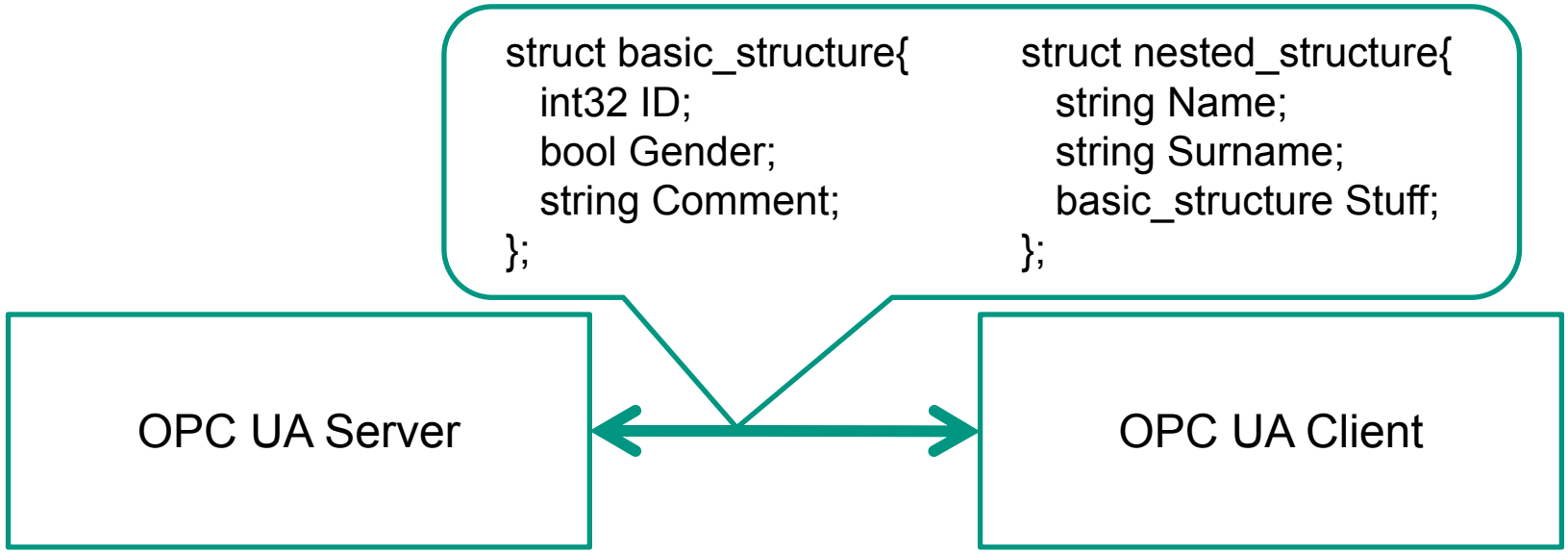
- OPC UA has a data type description system that works independent of the data encoding system, the encoding could be:
  - Binary
  - XML
  - JSON
- The UA python class should bind to the object in the figure so that after decoding a correspondent object can be delivered



* Detailed description can be found in the doc OPC Unified Architecture, Part 3, page 47

The OPC Binary Type Description System*

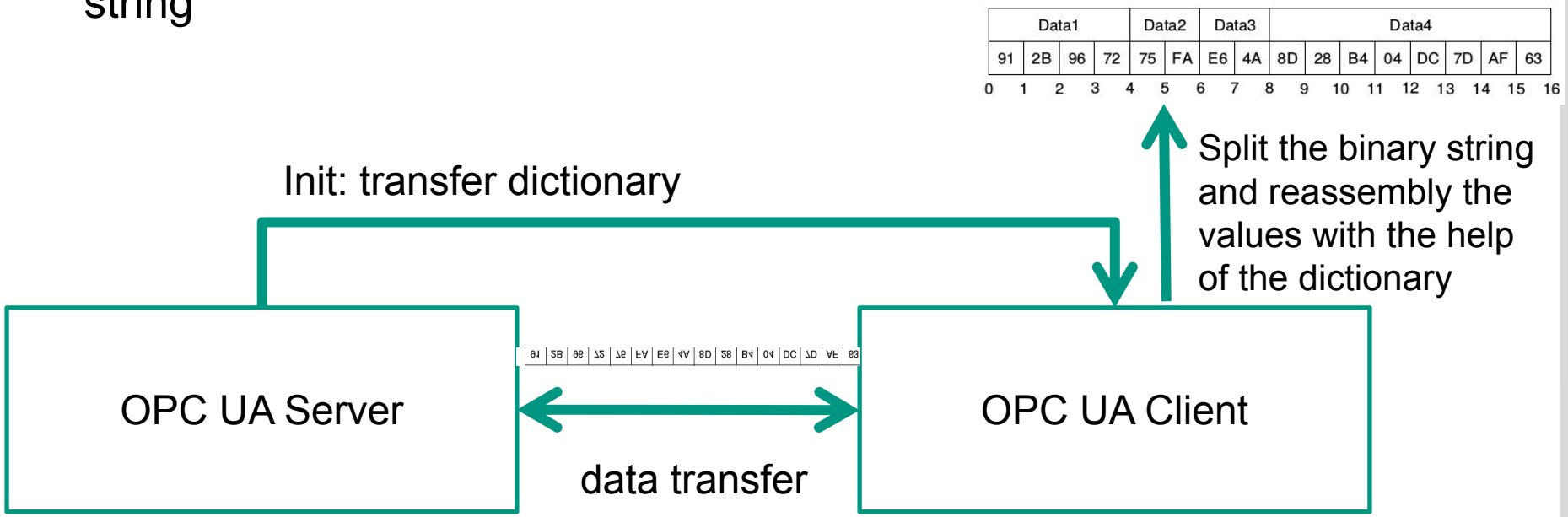
- Taken a concrete example to explain each component of the type description system, here the binary encoding system is used:
- Assume that we want to create two customized structures, and use them to communicate between a UA server and client.



* Detailed description can be found in the doc OPC Unified Architecture, Part 3, Annex C

The OPC Binary Type Description System*

- The transferred data is encoded as binary string
- The server should offer the client a type dictionary to let the client decode the customized data types
- According to the OPC UA standard, if a server passes a customized structure which the client cannot parse, it should be treated as a byte string



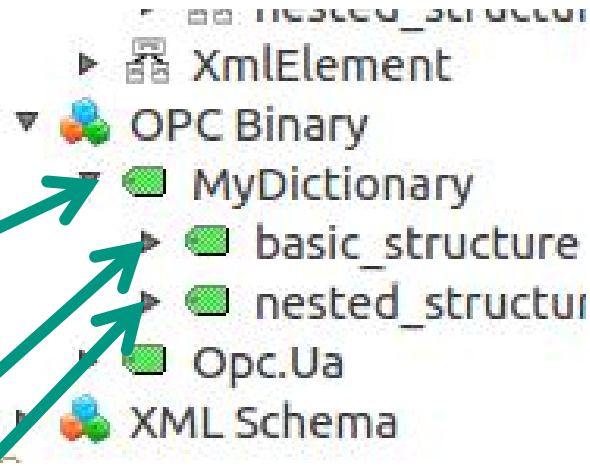
* Detailed description can be found in the doc OPC Unified Architecture, Part 3, Annex C

The OPC Binary Type Description System

- The dictionary consists of:
  - An item with type DataTypeDictionaryType, its value is a long byte string
  - Several items with the type DataTypeDescriptionType, their values are the keys

```
<opc:TypeDictionary DefaultByteOrder="LittleEndian"
TargetNamespace="http://examples.freeopcua.github.io"
xmlns:opc="http://opcfoundation.org/BinarySchema/"
xmlns:tns="http://examples.freeopcua.github.io" xmlns:ua="http://opcfoundation.org/UA/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <opc:Import Namespace="http://opcfoundation.org/UA/" />
  <opc:StructuredType BaseType="ua:ExtensionObject" Name="BasicStructure">
    <opc:Field Name="ID" TypeName="opc:Int32" />
    <opc:Field Name="Gender" TypeName="opc:Boolean" />
    <opc:Field Name="Comments" TypeName="opc:String" />
  </opc:StructuredType>
  <opc:StructuredType BaseType="ua:ExtensionObject" Name="NestedStructure">
    <opc:Field Name="Name" TypeName="opc:String" />
    <opc:Field Name="Surname" TypeName="opc:String" />
    <opc:Field Name="Stuff" TypeName="tns:BasicStructure" />
  </opc:StructuredType>
</opc:TypeDictionary>
```

As byte string



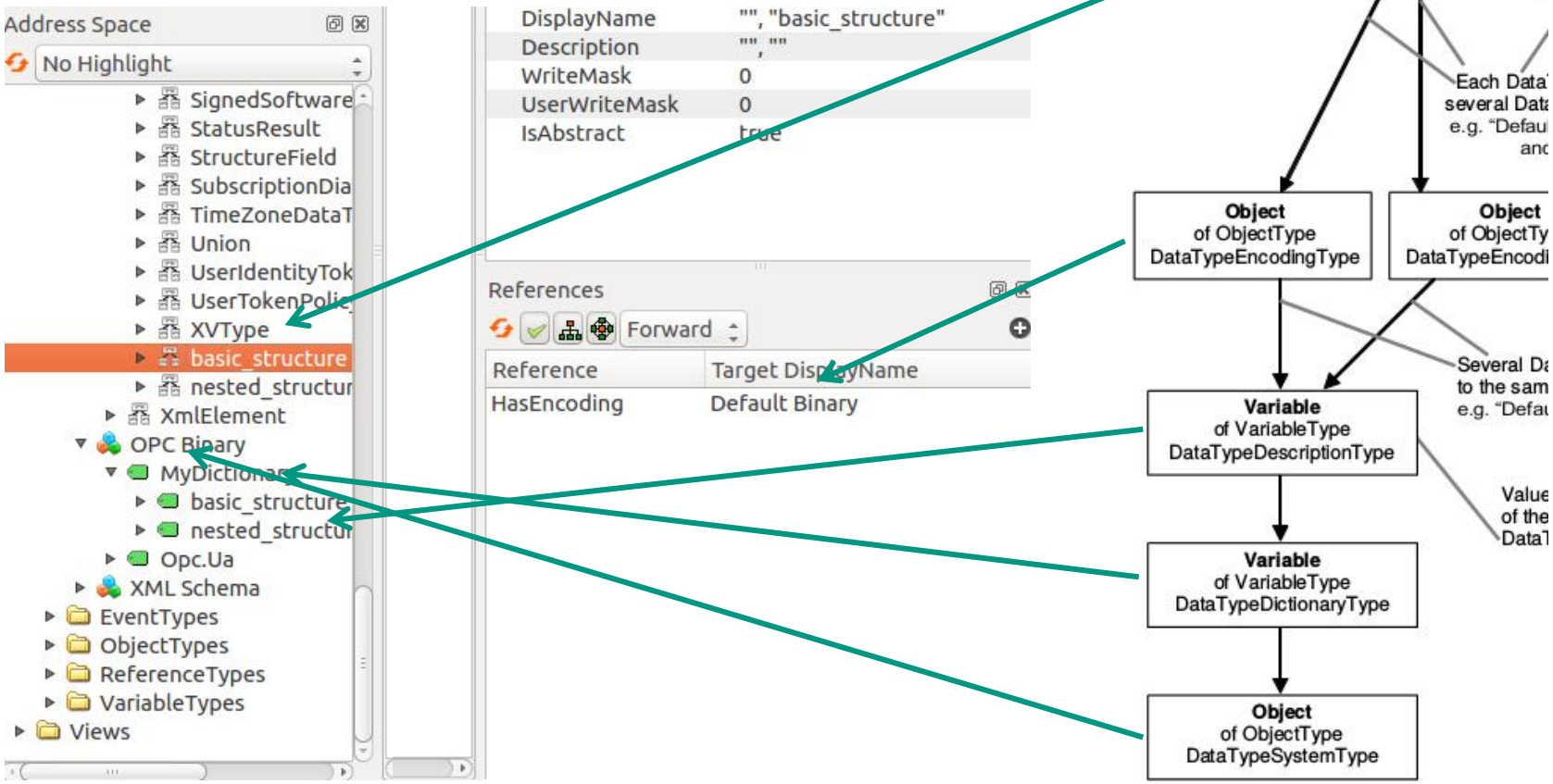
BasicStructure

NestedStructure

The OPC Binary Type Description System

Concrete use case

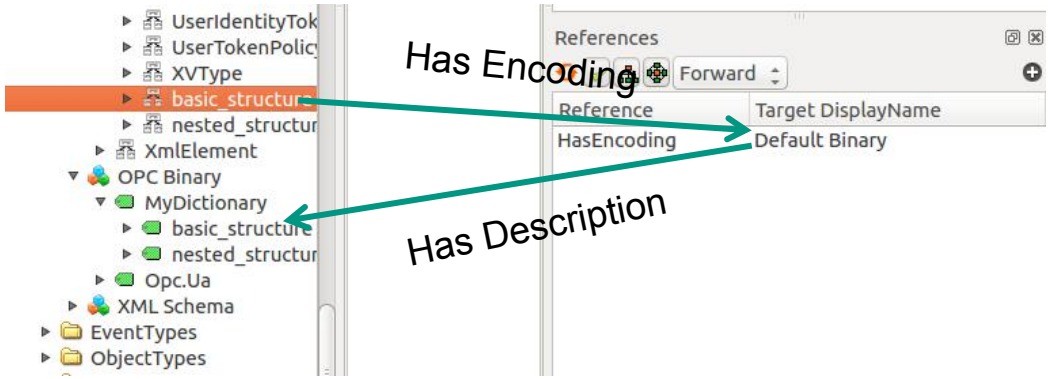
Concept



The OPC Binary Type Description System

- The programming tasks:
  - One Class creates the XML string according to the defined structure
  - One class creates the type dictionary, the correspondent UA nodes and their references

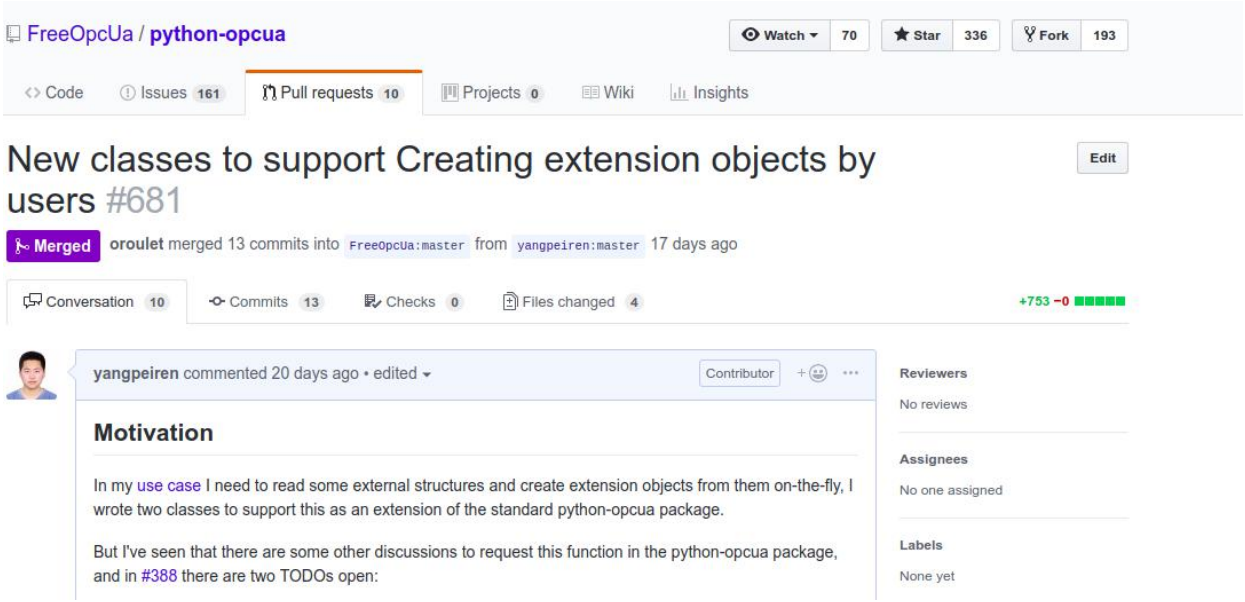
```
<opc:TypeDictionary DefaultByteOrder="LittleEndian"
TargetNamespace="http://examples.freeopcua.github.io"
xmlns:opc="http://opcfoundation.org/BinarySchema/"
xmlns:tns="http://examples.freeopcua.github.io" xmlns:ua="http://opcfoundation.org/UA/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <opc:Import Namespace="http://opcfoundation.org/UA/" />
  <opc:StructuredType BaseType="ua:ExtensionObject" Name="BasicStructure">
    <opc:Field Name="ID" TypeName="opc:Int32" />
    <opc:Field Name="Gender" TypeName="opc:Boolean" />
    <opc:Field Name="Comments" TypeName="opc:String" />
  </opc:StructuredType>
  <opc:StructuredType BaseType="ua:ExtensionObject" Name="NestedStructure">
    <opc:Field Name="Name" TypeName="opc:String" />
    <opc:Field Name="Surname" TypeName="opc:String" />
    <opc:Field Name="Stuff" TypeName="tns:BasicStructure" />
  </opc:StructuredType>
</opc:TypeDictionary>
```



The OPC Binary Type Description System

- The two classes realized customized structures (in OPC UA the type is named extension object)
- Since the code can not only be used in our project, it could also help other people with similar requirements, a pull request to the original repository of “python-opcua” was made, and it was merged into the project together with another bug fixing PR found while using the framework.

Please Reference the PR 679 and PR 681 for detailed description



Generate correspondent python UA classes

- Python classes should be generated from the customized structures both in server and client side to facilitate data transfer

```
<opc:TypeDictionary DefaultByteOrder="LittleEndian"
TargetNamespace="http://examples.freeopcua.github.io"
xmlns:opc="http://opcfoundation.org/BinarySchema/"
xmlns:tns="http://examples.freeopcua.github.io" xmlns:ua="http://opcfoundation.org/UA/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <opc:Import Namespace="http://opcfoundation.org/UA/" />
  <opc:StructuredType BaseType="ua:ExtensionObject" Name="BasicStructure">
    <opc:Field Name="ID" TypeName="opc:Int32" />
    <opc:Field Name="Gender" TypeName="opc:Boolean" />
    <opc:Field Name="Comments" TypeName="opc:String" />
  </opc:StructuredType>
  <opc:StructuredType BaseType="ua:ExtensionObject" Name="NestedStructure">
    <opc:Field Name="Name" TypeName="opc:String" />
    <opc:Field Name="Surname" TypeName="opc:String" />
    <opc:Field Name="Stuff" TypeName="tns:BasicStructure" />
  </opc:StructuredType>
</opc:TypeDictionary>
```

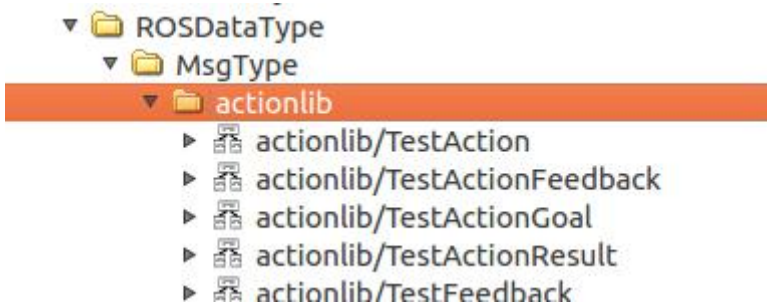
- This is a newly released function of the framework “python-opcua”, using the dictionary as reference to generate python code, then execute the code on-the-fly

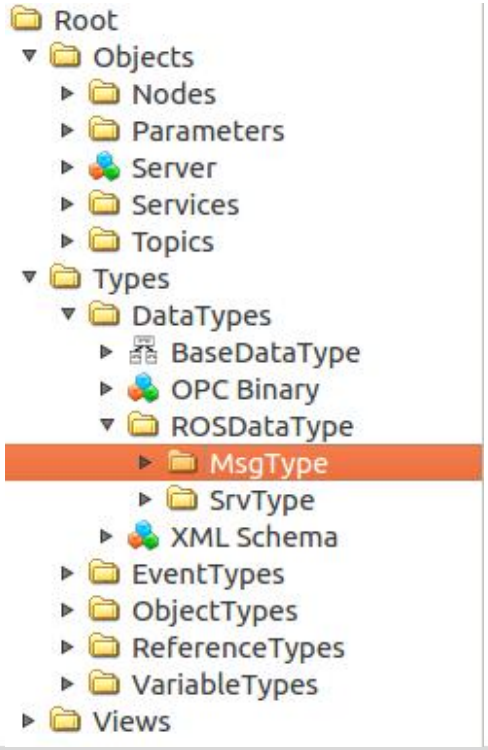
```
'''
THIS FILE IS AUTOGENERATED, DO NOT EDIT!!!
'''
from datetime import datetime
import uuid
from opcua import ua

class BasicStructure(object):

    ua_types = [
        ('ID', 'Int32'),
        ('Gender', 'Boolean'),
        ('Comments', 'String'),
    ]

    def __init__(self):
        self.ID = 0
        self.Gender = True
        self.Comments = None

class NestedStructure(object):

    ua_types = [
        ('Name', 'String'),
        ('Surname', 'String'),
        ('Stuff', 'BasicStructure'),
    ]

    def __init__(self):
        self.Name = None
        self.Surname = None
        self.Stuff = ua.BasicStructure()
```

New design based on the customized structures

- Modeling ROS static information
  - Retrieve the python classes of ROS Msgs and Srvs from roslib
  - Traverse the structure, generate customized structures
  - Generate python UA classes from the created XML string, therefore a one to one mapping of OPC UA class and ROS class is possible

- Display of ROS static information
  - Linked the data types in the ROSDataType folder
  - Msgs and Srvs are divided into two folders
  - Data types are organized by their libraries





New design based on the customized structures

- Modeling ROS dynamic information
  - ROS services
    - Service call, IO arguments are maximal one extension object respectively
  - ROS topics
    - Topic pub/sub, data objects are extension objects
  - ROS actions
    - Virtual link of the topics (soft reference type “Organizes”)
  - ROS nodes
    - Virtual link of the services, topics and actions relevant to one node
  - ROS parameters
    - Support not only create/delete but also value update
- ROSInfoAgent
  - “Eye” of the dynamic modeling
  - What is newly created, what is deleted in ROS will be retrieved only by this class, obey the design rule of modularity: few, small explicit interfaces (to roscore)

Hierachical structure of the project

- Evolution supporting design, future modification and extension of the project is easier

Layer	Correspondent files	Comment
Application	scripts/opcua_client_application_example.py, scripts/opcua_client_application_example_ros.py	possible application examples
Top	scripts/ros_server.py, scripts/ros_server_export_message.py, scripts/ros_opc_ua_client.py	create OPC UA server, call static and dynamic modeling, offer client support
High	scripts/ros_info_manage.py	created ROS information managers to manage the ROS services set, topic set, and ROS node set, aggregations are managed by one class
Middle	scripts/ros_opc_ua_comm.py other classes	build the communication proxy of ROS topics, services based on the UA objects
Low	scripts/ros_opc_ua_comm.py class OpcUaROSMessage	created correspondent OPC UA extension object of rosmsg, rossrv
Bottom	python-opcua/opcua/common/type_dictionary_buider.py	extension object support in python-opcua

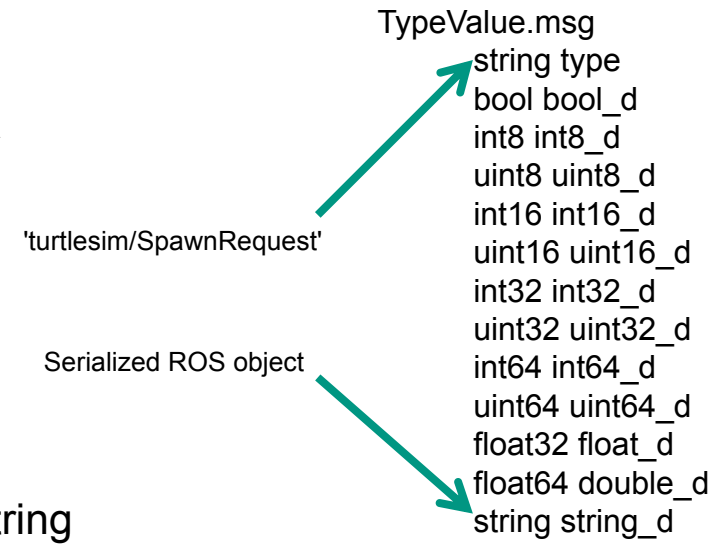
Advanced functions controled by parameters

- Some parameters are designed in the file “config/params.yaml”, which supports advanced functions, below is a list of them

Parameter name	Default value	Comment
namespace	'/'	filters of the ros topics, services, actions, parameters and nodes, default '/' means no filter
automatic_refresh	TRUE	True will start automatic refresh of ros nodes, otherwise the user should call refresh manually
refresh_cycle_time	0.5	The cycle time of the automatic refresh, the time unit is second
show_nodes	TRUE	Switch for masking the display of ROS nodes
show_topics	TRUE	Switch for masking the display of ROS topics
show_services	TRUE	Switch for masking the display of ROS services
show_params	TRUE	Switch for masking the display of ROS parameters
import_xml_msgs	FALSE	Controls if data types will be imported from an xml file or generate on-the-fly
parameter_writable	TRUE	Controls if the variables in parameters are writable

OPC UA Client as a ROS node

- A client that supports interacting with OPC UA server from ROS world, available services:
  - Connect/disconnect a server
  - List available UA nodes
  - Call UA method
  - Read/write a UA variable
  - Subscribe/unsubscribe a UA variable
- Pass arbitrary data types through the UA client to a server
  - type in the data type in the field “type”
  - Set the correspondent data field
  - A ROS object?
    - type in the class name in the field “type”
    - serialize the object, write the serialized string into the string field “string_d”



Summary

- Inherited the functionalities of the old version
- According to the OPC UA standard, implemented OPC UA extension objects
  - Merged the work into the open source project “python-opcua” to help more people with similar problems
  - Redesigned the modeling of ROS services, ROS topics, ROS nodes, ROS parameters based on the new implementation
- The new features of the current version
  - Performance improvement with less code
  - More functionalities
  - An envolving oriented structure

Karlsruhe Institute of Technology

IROB

Thanks for listening!

Any Questions?