DATEC AUTOMATION

tau4

Devel's Manual

Ver. 2017-06-04/1615

Table Of Contents

List Of Tables

List Of Figures

Revision History

Datum	Änderung
2016-04-19	Initial edition.

1 Glossary

tau4

Tools And Utilities. The '4' stands for 'for': tau for robotix, tau for math etc.

2 Introduction

TAU stands for Tools And Utilities. The question now is, should we stick with the name and especially with the concept of a package featuring each and everything?

3 TAUs Future

We could split TAU up into

ce

icom

ios

oop

pandora

tau4-stuff: data.flex

sensors

spy

tau4-stuff: logging, ThisName

wxs

wxPython support

stuff

Alternate names:

- **asot (All Sorts Of Things)**
- **aos (All Other Things)**
- ~~**ee (Everything Else)**~~
- **owt ('anything')**

4 tau4

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of tau4.
#
#   tau4 is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   tau4 is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with tau4. If not, see <http://www.gnu.org/licenses/>.


import copy
import sys
import uuid

#from _4all import _settings


__VERSION_NUMBER_MAJOR = 0
__VERSION_NUMBER_MINOR = 3
__VERSION_NUMBER_REV = 0


class _RevisionHistory:

    def __str__( self):
        return \
"""%s:
    2016-04-19:
        Created.

""" % __file__


class Object(object):

    def __init__( self, id):
        self.__id = id
        return

    def id( self):
        return self.__id


class _Objects:

    """Stores all Object instances.
    """

    def __init__( self):
        import threading

        self.__instances = {}
        self.__lock = threading.RLock()
        return

    def __call__( self, ident=None):
        with self.__lock:
            if ident is None:
                return self

            return self.__instances[ ident]

    def __contains__( self, ident):
        return ident in self.__instances

    def __len__( self):
        return len( self.__instances)

    def add( self, ident, instance):
        """Neues Objekt aufnehmen.

        \throws KeyError    wenn folgende Bedingungen **nicht** erfüllt sind:
                            \li **ident** ist eindeutig.

        \throws ValueError  wenn folgende Bedingungen **nicht** erfüllt sind:
                            \li **ident** ist ein String oder ein Integer\.
        """
        with self.__lock:
            if not isinstance( ident, (str, bytes, int)):
                raise ValueError( "instance.ident() must be a base string, but is a " + str( type( ident)))

            if ident in self.__instances:
                raise KeyError( "instance.ident() = '%s' isn't unique!" % ident)

            self.__instances[ ident] = instance
            return self

    def remove( self, ident):
        with self.__lock:
            del self.__instances[ ident]

    def lock( self):
        return self.__lock.acquire()

    def unlock( self):
        return self.__lock.release()


Objects = _Objects()


def ThisName( self=None, timestamp=False):
    """Name of the currently executed method or function.

    :param  self:   Instance of class of calling method.
                    Used to document the class the method belongs to.
    """
    level = 1
    if self is not None:
        if timestamp:
            return "[%f] %s::%s" % (time.time(), self.__class__.__name__, sys._getframe( level).f_code.co_name)

        return "%s::%s" % (self.__class__.__name__, sys._getframe( level).f_code.co_name)

    else:
        if timestamp:
            return "[%f] %s" % (time.time(), sys._getframe( level).f_code.co_name)

        return sys._getframe( level).f_code.co_name

    assert not "Trapped! "


class VersionInfo:

    """Revision history of all the tau4 packages found.
    """

    def __init__( self):
        self._version_tuple = ( __VERSION_NUMBER_MAJOR, __VERSION_NUMBER_MINOR, __VERSION_NUMBER_REV)
        return

    def __str__( self):
        return ".".join( self._version_tuple)

    def as_str( self):
        return str( self)

    def as_tuple( self):
        return self._version_tuple

    def changes( self):
        pn = "CHANGES.txt"
        try:
            f = open( pn, "rb")
            text = f.read()

        except IOError as e:
            text = u"Could not open file '%s' containing all changes of this version: '%s'" % (pn, e)

        return text

    def number_major( self):
        return self._version_tuple[ 0]

    def number_minor( self):
        return self._version_tuple[ 1]

    def number_revision( self):
        return self._version_tuple[ 2]
```

5 automation

This module implements stuff useful in automation projects.

5.1 ces.py – Control Engineering Systems

5.1.1 Source Code

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of tau4.
#
#   tau4 is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   tau4 is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with tau4. If not, see <http://www.gnu.org/licenses/>.


import logging; _Logger = logging.getLogger()

import abc, time

from tau4.data import flex
from tau4.sweng import overrides, PublisherChannel
from tau4.threads import Cycler


class Node4C(metaclass=abc.ABCMeta):

    """

    .. note::
        Es ist die Frage, ob ein _Node einen Eingang und einen Ausgang braucht,
        denn das kann mit den Variablen erledigt werden, von denen gelesen und
        auf die geschrieben wird und die ja dem Ctor übergeben werden.
        Schließlich sind alle Subclasses von _Node appspez. und damit weiß die
        App, wie die Nodes über welche Variablen zusammenhängen müssen.
    """

    def __init__( self):
        self.__node_next = None
        self.__is_on = False
        self.__is_running = False
        return

    @abc.abstractmethod
    def configure( self, fv_Ts):
        """Konfigurieren des Nodes.

        Beispielsweise wird der Algorithmus hier die Koeffizienten der
        Differenzengleichung berechnen wollen.

        .. caution::
            Diese Methode muss am Ende von configure() jeder abgeleiteten Klasse
            aufgerufen werden - **zwingend**!

        :param  FlexVarblLL fv_Ts:  Abtastzeit.
        """
        if self.__node_next:
            self.__node_next.configure( fv_Ts)

        return self

    @abc.abstractmethod
    def execute( self):
        """Ausführen des Nodes.

        .. caution::
            Diese Methode muss am Ende von execute() jeder abgeleiteten Klasse
            aufgerufen werden - **zwingend**!
        """
        if self.__node_next:
            self.__node_next.execute()
        return self

    def is_on( self):
        """Es werden nur von der Hardware eingelesene Werte angezeigt, der Algorithmus wird nicht ausgeführt.
       """
        return self.__is_on

    def is_ready( self):
        """Es werden nur von der Hardware eingelesene Werte angezeigt, der Algorithmus wird nicht ausgeführt.
        """
        return self.__is_on and not self.__is_running

    def is_running( self):
        """Es werden nur von der Hardware eingelesene sowie berechnete Werte angezeigt, **der Algorithmus wird also 
ausgeführt**.
       """
        return self.__is_running

    def node_last( self):
        """Liefert den letzten Node der Kette, zu der dieser Node gehört.
        """
        node = self
        while node:
            if not node.node_next():
                return node

            node = node.node_next()

        return None

    def node_next( self, node=None):
        """Next node in the linked list.
        """
        if node is None:
            return self.__node_next

        self.__node_next = node
        return self

    def to_off( self):
        """Switch OFF Node.
        """
        self.__is_on = False
        if self.__node_next:
            self.__node_next.to_off()

        return

    def to_on( self):
        """Switch ON Node.
        """
        self.__is_on = True
        if self.__node_next:
            self.__node_next.to_on()

        return

    def to_ready( self):
        """Switch from RUNNING to READY.
        """
        self.__is_running = False
        if self.__node_next:
            self.__node_next.to_ready()

        return

    def to_running( self):
        """Switch to RUNNING.

        Node must be ON already.
        """
        if self.is_on():
            self.__is_running = True

        if self.__node_next:
            self.__node_next.to_running()

        return


class NodeReconfigurator(Node4C):

    """Konfigurations-Node.

    Er veranlasst z.B. die Neu/berechnung der
    Koeffizenten der Differenzengleichung.

    Mit diesem Knoten beginnt die Linked List, die den Regler ausmacht,
    **automatsich**, d.h. dieser Node muss vom User nicht eingehängt werden.
    """

    def __init__( self, fv_Ts):
        super().__init__()
        fv_Ts.reg_tau4s_on_modified( self._tau4s_on_Ts_modified_)
        self.__is_dirty = True
                                        # Bei Erstausführung muss auf
                                        #   jeden Fall eine Konfiguration
                                        #   erfolgen.
        self.__fv_Ts = fv_Ts
        return

    def configure( self, fv_Ts):
        if self.__is_dirty:
            self.__is_dirty = False
            super().configure( self.__fv_Ts)

        return self


    def execute( self):
        self.configure( self.__fv_Ts)

        super().execute()
        return self

    def _tau4s_on_Ts_modified_( self, tau4pc):
        self.__Ts = tau4pc.client().fv_Ts.value()
        self.__is_dirty = True
        return self


class SISOController:

    """Container für die Nodes, aus denen der Regler besteht.

    Im folgenden ein Beispiel, wie die Konstruktion eines Reglers aussehen kann.

    Usage::

        ##  Default-Parameter für die Regler holen
        #
        fv_Kp = flex.Variable( value=1.0)
        fv_Ki = flex.Variable( value=1.0)
        fv_Kd = flex.Variable( value=1.0)
        fv_alpha = flex.Variable( value=0.7)

        ##  Variable holen, über die die Nodes zusammenhängen und mit
        #   denen sie arbeiten.
        #
        fv_w = flex.Variable( value=100.0)
        fv_y = flex.Variable( value=0.0)
        fv_e = flex.Variable( value=0.0)
        fv_u = flex.VariableMo( id=-1, value=0.0, value_min=-400, value_max=400)
                                        # fv_u als VariableMo def'en, damit
                                        #   Sättigung entdeckt werden kann.
                                        #   Die  B e r e i c h s g r e n z e n
                                        #   m ü s s e n   mit dem Aktuator-Knoten
                                        #   a b g e s t i m m t   werden!
        fv_Ts = flex.VariableDeClMoPe( id="controller.Ts", label="Ts", dim="s", value=0.010, value_min=0.001,
value_max=None)

        ##  Variable fürs GUI holen
        #
        fv = flex.VariableDeClMo( id="gui.w(t)", label="u(2)", dim="", value=0.0, value_min=None, value_max=None)
        flex.Variable.InstanceStore( fv.id(), fv)
        fv.reg_tau4s_on_modified( self._controller_data_changed_)
        fv = flex.VariableDeClMo( id="gui.y(t)", label="u(2)", dim="", value=0.0, value_min=None, value_max=None)
        flex.Variable.InstanceStore( fv.id(), fv)
        fv.reg_tau4s_on_modified( self._controller_data_changed_)
        fv = flex.VariableDeClMo( id="gui.e(t)", label="u(2)", dim="", value=0.0, value_min=None, value_max=None)
        flex.Variable.InstanceStore( fv.id(), fv)
        fv.reg_tau4s_on_modified( self._controller_data_changed_)
        fv = flex.VariableDeClMo( id="gui.u(t)", label="u(2)", dim="", value=0.0, value_min=None, value_max=None)
        flex.Variable.InstanceStore( fv.id(), fv)
        fv.reg_tau4s_on_modified( self._controller_data_changed_)

        ##  Algorithmen erzeugen für die anschließenden Tests
        #
        algorithms = []
        algorithms.append( EulerBw4P( id=id, fv_Kp=fv_Kp, fv_Ts=fv_Ts, fv_e=fv_e, fv_u=fv_u))
        algorithms.append( EulerBw4PDT1( id=-1, fv_Kp=fv_Kp, fv_Kd=fv_Kd, fv_alpha=fv_alpha, fv_e=fv_e, fv_u=fv_u,
fv_Ts=fv_Ts))
        algorithms.append( EulerBw4PIDT1( id=id, fv_Kp=fv_Kp, fv_Ki=flex.Variable( value=0.0), fv_Kd=fv_Kd,
fv_alpha=fv_alpha, fv_Ts=fv_Ts, fv_e=fv_e, fv_u=fv_u))
        algorithms.append( EulerBw4PIDT1p( id=-1, fv_Kp=fv_Kp, fv_Ki=flex.Variable( value=0.0), fv_Kd=fv_Kd,
fv_alpha=fv_alpha, fv_e=fv_e, fv_u=fv_u, fv_Ts=fv_Ts))
        algorithms.append( EulerBw4PIDT1( id=id, fv_Kp=fv_Kp, fv_Ki=fv_Ki, fv_Kd=fv_Kd, fv_alpha=fv_alpha,
fv_Ts=fv_Ts, fv_e=fv_e, fv_u=fv_u))
        algorithms.append( EulerBw4PIDT1p( id=-1, fv_Kp=fv_Kp, fv_Ki=fv_Ki, fv_Kd=fv_Kd, fv_alpha=fv_alpha, fv_e=fv_e,
fv_u=fv_u, fv_Ts=fv_Ts))

        from matplotlib.backends.backend_pdf import PdfPages
        pdfpages = PdfPages( "NodePrinter-printed_figures.pdf")
        for algorithm in algorithms:

            controller = SISOController.New(
                (
                    NodeSummingPoint( fv_w, fv_y, fv_e),
                    NodeAlgorithm( algorithm=algorithm),
                    NodeActuator( fv_u),
                    NodePublisher( fv_w=fv_w, fv_y=fv_y, fv_e=fv_e, fv_u=fv_u),
                    NodePrinter( fv_w=fv_w, fv_y=fv_y, fv_e=fv_e, fv_u=fv_u, fv_Ts=fv_Ts, algorithm=algorithm,
pdfpages=pdfpages)
                ),
                fv_Ts
            )

            controller.to_on()
                                            # Controller ist READY
            controller.to_running()
                                            # Controller läuft
            rectangle = Signals.RECTANGLE( 100, 0.1)
            for i in range( 1000):
                fv_w.value( rectangle( i*fv_Ts.value()))
                controller.execute()
                print( "Runtime consumption = %.3f ms. " % (controller.runtime() * 1000))

            controller.to_ready()
                                            # Controller ist wieder nur READY
            controller.to_off()

        pdfpages.close()
        return

    .. todo::
        Controller von :py:class:`Node4C` ableiten?
    """

    @staticmethod
    def New( nodes, fv_Ts):
        node1 = NodeReconfigurator( fv_Ts)
                                        # Konfigurations-Node. Er veranlasst z.B. die Neu/berechnung der
                                        #   Koeffizenten der Differenzengleichung.
                                        #
                                        #   Mit diesem Knoten begint die Linked List, die den Regler ausmacht.
                                        #
        for node in nodes:
            node1.node_last().node_next( node)

        controller = SISOController( node1)
                                        # Es fällt auf, dass dem Regler die
                                        #   Abtastzeit nicht übergeben wird, die
                                        #   für die Berechnung der Koeffizienten
                                        #   der Differenzengleichung benötigt wird.
                                        #   Da diese Berechnung in der Methode
                                        #   configure() erfolgt, wird dort die
                                        #   Abtastzeit und nur die Abtastzeit
                                        #   übergeben.
        return controller

    def __init__( self, node1):
        self.__node1 = node1

        self.__is_on = False
        self.__is_ready = False

        self.__runtime = 0
        return

    def execute( self):
        """Regler ausführen.
        """
        dt = time.time()
        if self.__node1:
            self.__node1.execute()

        dt = time.time() - dt
        self.__runtime = dt
        return

    def is_on( self):
        """Es werden nur von der Hardware eingelesene Werte angezeigt, der Algorithmus wird nicht ausgeführt.
        """
        return self.__is_on

    def is_ready( self):
        """Es werden nur von der Hardware eingelesene Werte angezeigt, der Algorithmus wird nicht ausgeführt.
        """
        return self.__is_on and not self.__is_running

    def is_running( self):
        """Es werden nur von der Hardware eingelesene sowie berechnete Werte angezeigt, **der Algorithmus wird also
ausgeführt**.
        """
        return self.__is_running

    def runtime( self):
        """Laufzeitbedarf für execute().
        """
        return self.__runtime

    def to_off( self):
        """Regler ausschalten.
        """
        self.__is_on = False
        self.__node1.to_off()
        return

    def to_on( self):
        """Regler einschalten.
        Nodes müssen so implementiert werden, dass zwar keine Einflussnahme auf
        die Strecke erfolgt, weiterhin aber Werte angezeigt werden können.
        """
        self.__is_on = True
        self.__node1.to_on()
        return

    def to_ready( self):
        """Regler von RUNNING auf READY schalten.

        Nodes müssen so implementiert werden, dass zwar keine Einflussnahme auf
        die Strecke erfolgt, weiterhin aber Werte angezeigt werden können.
        """
        self.__is_running = False
        self.__node1.to_ready()
        return

    def to_running( self):
        """Regler auf RUNNING schalten.
        """
        if self.is_on():
            self.__is_running = True

        self.__node1.to_running()
        return
```

5.2 plc.py – PLC

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of tau4.
#
#   tau4 is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   tau4 is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with tau4. If not, see <http://www.gnu.org/licenses/>.


import logging; _Logger = logging.getLogger()

import abc
from collections import OrderedDict
import time

from tau4.automation.sm import SM, SMState
from tau4.data import flex
from tau4.io.hal.io import IOSpecPool
from tau4.sweng import overrides, PublisherChannel, Singleton
from tau4.threads import Cycler


class OpMode:

    class _Common(metaclass=Singleton):

        __switch_AUTO = False
        __switch_EMSTOP = False
        __switch_ENABLE = False
        __switch_ON = False
        __button_START = False
        __button_STOP = False


        def button_START( self, arg=None):
            if arg is None:
                return self.__button_START

            self.__button_START = arg
            return self

        def button_STOP( self, arg=None):
            if arg is None:
                return self.__button_STOP

            self.__button_STOP = arg
            return self

        def switch_AUTO( self, arg=None):
            if arg is None:
                return self.__switch_AUTO

            self.__switch_AUTO = arg
            return self

        def switch_EMSTOP( self, arg=None):
            if arg is None:
                return self.__switch_EMSTOP

            self.__switch_EMSTOP = arg
            return self

        def switch_ENABLE( self, arg=None):
            if arg is None:
                return self.__switch_ENABLE

            self.__switch_ENABLE = arg
            return self

        def switch_ON( self, arg=None):
            if arg is None:
                return self.__switch_ON

            self.__switch_ON = arg
            return self

        def switch_EMSTOP( self, arg=None):
            if arg is None:
                return self.__switch_EMSTOP

            self.__switch_EMSTOP = arg
            return self


    class EmStop(SMState):

        def condition_EMSTOP_IS_RELEASED( self):
            return not self.common().switch_EMSTOP()

        @overrides( SMState)
        def execute( self):
            super().execute()   # Nur zum Breakpoint-Setzen
            return

        @overrides( SMState)
        def value( self):
            return OpModeValues._EMSTOP


    class Off(SMState):

        def condition__SWITCH_ON_IS_ON( self):
            return self.common().switch_ON()

        def condition__EMSTOP_IS_PRESSED( self):
            return self.common().switch_EMSTOP()

        @overrides( SMState)
        def execute( self):
            super().execute()   # Nur zum Breakpoint-Setzen
            return

        @overrides( SMState)
        def value( self):
            return OpModeValues._OFF


    class On(SMState):

        def condition__SWITCH_AUTO_IS_OFF( self):
            return not self.common().switch_AUTO()

        def condition__SWITCH_AUTO_IS_ON( self):
            return self.common().switch_AUTO()

        def condition__SWITCH_ON_IS_OFF( self):
            return not self.common().switch_ON()

        def condition__EMSTOP_IS_PRESSED( self):
            return self.common().switch_EMSTOP()

        @overrides( SMState)
        def execute( self):
            super().execute()   # Nur zum Breakpoint-Setzen
            return

        @overrides( SMState)
        def value( self):
            return OpModeValues._ON


    class Manu:
        class Idle(SMState):

            def condition__SWITCH_AUTO_IS_OFF( self):
                return not self.common().switch_AUTO()

            def condition__SWITCH_AUTO_IS_ON( self):
                return self.common().switch_AUTO()

            def condition__SWITCH_ENABLE_IS_ON( self):
                return self.common().switch_ENABLE()

            def condition__SWITCH_ON_IS_OFF( self):
                return not self.common().switch_ON()

            def condition__EMSTOP_IS_PRESSED( self):
                return self.common().switch_EMSTOP()

            @overrides( SMState)
            def execute( self):
                super().execute()   # Nur zum Breakpoint-Setzen
                return

            @overrides( SMState)
            def value( self):
                return OpModeValues.MANU._IDLE


        class Ready(SMState):

            def condition__SWITCH_AUTO_IS_OFF( self):
                return not self.common().switch_AUTO()

            def condition__SWITCH_AUTO_IS_ON( self):
                return self.common().switch_AUTO()

            def condition__SWITCH_ENABLE_IS_OFF( self):
                return not self.common().switch_ENABLE()

            def condition__SWITCH_ON_IS_OFF( self):
                return not self.common().switch_ON()

            def condition__EMSTOP_IS_PRESSED( self):
                return self.common().switch_EMSTOP()

            @overrides( SMState)
            def execute( self):
                super().execute()   # Nur zum Breakpoint-Setzen
                return

            @overrides( SMState)
            def value( self):
                return OpModeValues.MANU._READY


    class Auto:

        class Idle(SMState):

            def condition__SWITCH_AUTO_IS_OFF( self):
                return not self.common().switch_AUTO()

            def condition__SWITCH_ENABLE_IS_OFF( self):
                return not self.common().switch_ENABLE()

            def condition__SWITCH_ENABLE_IS_ON( self):
                return self.common().switch_ENABLE()

            def condition__SWITCH_ON_IS_OFF( self):
                return not self.common().switch_ON()

            def condition__EMSTOP_IS_PRESSED( self):
                return self.common().switch_EMSTOP()

            @overrides( SMState)
            def execute( self):
                super().execute()   # Nur zum Breakpoint-Setzen
                return

            @overrides( SMState)
            def value( self):
                return OpModeValues.AUTO._IDLE


        class Ready(SMState):

            def close( self):
                if self.common().button_START():
                    self.common().button_START( False)

                super().close()
                return self

            def condition__BUTTON_START_IS_PRESSED( self):
                return self.common().button_START()

            def condition__SWITCH_AUTO_IS_OFF( self):
                return not self.common().switch_AUTO()
            def condition__SWITCH_AUTO_IS_ON( self):
                return self.common().switch_AUTO()

            def condition__SWITCH_ENABLE_IS_OFF( self):
                return not self.common().switch_ENABLE()

            def condition__SWITCH_ON_IS_OFF( self):
                return not self.common().switch_ON()

            def condition__EMSTOP_IS_PRESSED( self):
                return self.common().switch_EMSTOP()

            @overrides( SMState)
            def execute( self):
                super().execute()   # Nur zum Breakpoint-Setzen
                return

            @overrides( SMState)
            def value( self):
                return OpModeValues.AUTO._READY


        class Running(SMState):

            def close( self):
                if self.common().button_STOP():
                    self.common().button_STOP( False)

                super().close()
                return self

            def condition__BUTTON_STOP_IS_PRESSED( self):
                return self.common().button_STOP()

            def condition__SWITCH_AUTO_IS_OFF( self):
                return not self.common().switch_AUTO()

            def condition__SWITCH_AUTO_IS_ON( self):
                return self.common().switch_AUTO()

            def condition__SWITCH_ENABLE_IS_OFF( self):
                return not self.common().switch_ENABLE()

            def condition__SWITCH_ON_IS_OFF( self):
                return not self.common().switch_ON()

            def condition__EMSTOP_IS_PRESSED( self):
                return self.common().switch_EMSTOP()

            @overrides( SMState)
            def execute( self):
                super().execute()   # Nur zum Breakpoint-Setzen
                return

            @overrides( SMState)
            def value( self):
                return OpModeValues.AUTO._RUNNING



    def __init__( self):
        _SMSTable = {\
            self.Off(): \
                [ \
                    (self.Off().condition__SWITCH_ON_IS_ON, self.On()),
                    (self.Off().condition__EMSTOP_IS_PRESSED, self.EmStop()),
                ],

            self.On():
                [ \
                    (self.On().condition__SWITCH_AUTO_IS_ON, self.Auto.Idle()),
                    (self.On().condition__SWITCH_AUTO_IS_OFF, self.Manu.Idle()),
                ],

            self.Manu.Idle():
                [ \
                    (self.Manu.Idle().condition__SWITCH_AUTO_IS_ON, self.Auto.Idle()),
                    (self.Manu.Idle().condition__SWITCH_ENABLE_IS_ON, OpMode.Manu.Ready()),
                    (self.Manu.Idle().condition__SWITCH_ON_IS_OFF, self.Off()),
                ],

            OpMode.Manu.Ready():
                [ \
                    (OpMode.Manu.Ready().condition__SWITCH_AUTO_IS_ON, self.Manu.Idle()),
                    (OpMode.Manu.Ready().condition__SWITCH_ENABLE_IS_OFF, self.Manu.Idle()),
                    (OpMode.Manu.Ready().condition__SWITCH_ON_IS_OFF, self.Off()),
                ],

            OpMode.Auto.Idle():
                [ \
                    (OpMode.Auto.Idle().condition__SWITCH_ON_IS_OFF, OpMode.Off()),
                    (OpMode.Auto.Idle().condition__SWITCH_AUTO_IS_OFF, OpMode.Manu.Idle()),
                    (OpMode.Auto.Idle().condition__SWITCH_ENABLE_IS_ON, OpMode.Auto.Ready()),
                ],
            OpMode.Auto.Ready():
                [ \
                    (OpMode.Auto.Ready().condition__SWITCH_AUTO_IS_OFF, self.Auto.Idle()),
                    (OpMode.Auto.Idle().condition__SWITCH_ENABLE_IS_OFF, self.Auto.Idle()),
                    (OpMode.Auto.Ready().condition__SWITCH_ON_IS_OFF, self.Off()),
                    (OpMode.Auto.Ready().condition__BUTTON_START_IS_PRESSED, OpMode.Auto.Running()),
                ],

            OpMode.Auto.Running():
                [ \
                    (OpMode.Auto.Running().condition__SWITCH_AUTO_IS_OFF, OpMode.Auto.Ready()),
                    (OpMode.Auto.Running().condition__SWITCH_ENABLE_IS_OFF, OpMode.Auto.Ready()),
                    (OpMode.Auto.Running().condition__SWITCH_ON_IS_OFF, OpMode.Off()),
                    (OpMode.Auto.Running().condition__BUTTON_STOP_IS_PRESSED, OpMode.Auto.Ready()),
                ],

            OpMode.EmStop(): \
                [ \
                    (OpMode.EmStop().condition_EMSTOP_IS_RELEASED, OpMode.Off()),
                ],

        }

        self.__sm = SM( _SMSTable, self.Off(), self._Common())
        return

    ### Query the actual operation mode
    #
    def is_AUTO( self):
        return self.sm().smstate_current() is self.Auto()

    def is_AUTO_AND_READY( self):
        return self.sm().smstate_current() is OpMode.Auto.Ready()

    def is_AUTO_AND_RUNNING( self):
        return self.sm().smstate_current() is OpMode.Auto.Running()

    def is_OFF( self):
        return self.sm().smstate_current() is self.Off()

    def is_MANU( self):
        return self.sm().smstate_current() is self.Manu()

    def is_MANU_AND_READY( self):
        return self.sm().smstate_current() is OpMode.Manu.Ready()

    def is_ON( self):
        return self.sm().smstate_current() is self.On()

    ### Change the actual operation mode
    #
    def button_START( self, arg):
        self.sm().common().button_START( arg)
        return self

    def button_STOP( self, arg):
        self.sm().common().button_STOP( arg)
        return self

    def switch_AUTO( self, arg):
        self.sm().common().switch_AUTO( arg)
        return self

    def switch_ENABLE( self, arg):
        self.sm().common().switch_ENABLE( arg)
        return self

    def switch_ON( self, arg):
        self.sm().common().switch_ON( arg)
        return self

    def switch_EMSTOP( self, arg):
        self.sm().common().switch_EMSTOP( arg)
        return self
    ### Some queries besicdes the one about the iperation mode
    #
    def name( self):
        return self.sm().fv_smstatename_current().value()

    def sm( self):
        return self.__sm


class OpModeValues:

    _OFF = 0
    _ON = 1

    class MANU:

        _IDLE = 10
        _READY = 11


    class AUTO:
        _IDLE = 20
        _READY = 21
        _RUNNING = 22

    _EMSTOP = 9


class OpModeNameFinder(metaclass=Singleton):

    def name( self, value):
        return \
            {
                OpModeValues._OFF: "OFF",
                OpModeValues._ON: "ON",
                OpModeValues._EMSTOP: "EMSTOP",
                OpModeValues.MANU._IDLE: "MANU - IDLE",
                OpModeValues.MANU._READY: "MANU - READY",
                OpModeValues.AUTO._IDLE: "AUTO - IDLE",
                OpModeValues.AUTO._READY: "AUTO - READY",
                OpModeValues.AUTO._RUNNING: "AUTO - RUNNING",

            }[ value]


class PLC(Cycler, metaclass=abc.ABCMeta):

    """SPS.

    Eine SPS ist üblicherweise der "Schrittmacher" in Automatisierungslösungen.
    """

    def __init__( self, *, cycletime_plc, cycletime_ios, is_daemon, startdelay=0):
        """

        :param  iainps:
            Appspez. interne I/Os. Müssen Hier angegeben werden, damit sie zur
            richtigen Zeit execute()ed werden können. Definition und Ausführung
            liegen völlig im Einflussbereich der App.

        Usage::
            2DO: Code aus iio hier her kopieren.
        """
        super().__init__( cycletime=cycletime_plc, udata=None, is_daemon=is_daemon)

        self.__jobs = []
        self.__jobindexes = dict( list( zip( [ job.id() for job in self.__jobs], list( range( len( self.__jobs))))))

        self.__operationmode = OpMode()
        return

    def _inps_execute_( self):
        IOSpecPool().execute_inps()
        return

    def _outs_execute_( self):
        IOSpecPool().execute_outs()
        return

    @abc.abstractmethod
    def _iinps_execute_( self):
        """Interne Inputs lesen.

        Beispiel::
            @overrides( automation.PLC)
            def _iinps_execute_( self):
                iIOs().idinps_plc().execute()
                return
        """
        pass

    @abc.abstractmethod
    def _iouts_execute_( self):
        """Interne Outouts schreiben.

        Beispiel::
            @overrides( automation.PLC)
            def _iouts_execute_( self):
                iIOs().idouts_plc().execute()
                return
        """
        pass

    def job_add( self, job):
        if job.cycletime() < self.cycletime():
            raise ValueError( "Cycletime of Job '%s' must not be greater than %f, but is %f!" %
(self.__class__.__name__, self.cycletime(), job.cycletime()))

        self.__jobs.append( job)
        self.__jobindexes = dict( list( zip( [ job.id() for job in self.__jobs], list( range( len( self.__jobs))))))
        return self

    def jobs( self, id=None):
        if id is None:
            return self.__jobs

        return self.__jobs[ self.__jobindexes[ id]]
    def operationmode( self) -> OpMode:
        return self.__operationmode

    def _run_( self, udata):
        ### Alle Eingänge lesen
        #
        self._inps_execute_()

        ### Alle internen Eingänge lesen
        #
        self._iinps_execute_()

        ### Jobs ausführen, wobei der erste Job immer das Lesen der INPs und
        #   der letzte Job das Schreiben der OUTs ist.
        #
        self.operationmode().sm().execute()

        jobs2exec = []
        for job in self.__jobs:
            job.rtime_decr( self.cycletime())
            if job.rtime() <= 0:
                jobs2exec.append( job)
                job.rtime_reset()

        for job in jobs2exec:
            job.execute()

        ### Alle Ausgänge schreiben
        #
        self._outs_execute_()

        ### Alle internen Ausgänge schreien
        #
        self._iouts_execute_()

        return


class Job(metaclass=abc.ABCMeta):

    """Job, der innerhalb eines Zyklus auszuführen ist.

    Alle Jobs werden innerhalb ein und desselben Zyklus ausgeführt.

    **Zugriff auf die I/Os**:
        Die I/Os werden vo der PLC geselsen und geschrieben, sodass jeder
        Job auf die mit dem I/O verbundene Variable zugreifen muss/darf.

        **Beispiel**::
            ### Eingang lesen
            #
            if HAL4IOs().dinps( 7).fv_raw().value():
                DO_THIS()

            else:
                DO_THAT()

            ### Ausgang schreiben
            #
            HAL4IOs().douts( 7).fv_raw().value( 1)

    Ein Job kann alles Mögliche sein. Hier ein paar Beispiele:

    -   Schlüsselschalter lesen und Betriebsart umschalten, je nach Schlüsselschalterstellung.
    """

    _CYCLE_TIME = 0.050

    def __init__( self, plc, id, cycletime):
        self.__plc = plc
        self.__id = id if not id in (-1, None, "") else self.__class__.__name__
        self.__cycletime = cycletime
        self.__time_rem = cycletime

        return

    def cycletime( self):
        return self.__cycletime

    @abc.abstractmethod
    def execute( self):
        pass

    def id( self):
        return self.__id

    def plc( self) -> PLC:
        return self.__plc

    def rtime_decr( self, secs):
        """Decrease the remaining time until executed.
        """
        self.__time_rem -= secs
        return self

    def rtime( self):
        """Remaining time until executed.
        """
        return self.__time_rem

    def rtime_reset( self):
        """Reset remaining time until executed to cycle time again.
        """
        self.__time_rem = self.__cycletime
        return self
```

5.3 sm.py – State Machines

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of tau4.
#
#   tau4 is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   tau4 is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with tau4. If not, see <http://www.gnu.org/licenses/>.

import abc

from tau4 import ThisName
from tau4.data import flex
from tau4.datalogging import UsrEventLog
from tau4.sweng import PublisherChannel, Singleton


class SM:

    """State Machine.

    An app can have more than one state machine.
    But be aware, that state machine states are singletons and you have to decide at
    runtime, which state machine they belong to!

    Usage:

        self.__sm = _SMStates4ROP().sm()

        class _SMStates4ROP:

            ############################################################################
            ### States As Local Classes
            #
            class _SMState(SMState):

                def close( self):
                    super().close()

                    UsrEventLog().log_info( "Leaving state '%s'. " % self.name(), ThisName( self))
                    return

                def exitcondition_ROPDISABLE_IS_REQUESTED( self):
                    return iIOs().idinps_rop().idinp_ROP_ENABLE().value() == 0

                def exitcondition_ROPENABLE_IS_REQUESTED( self):
                    return iIOs().idinps_rop().idinp_ROP_ENABLE().value() == 1

                def exitcondition_ROPSTART_IS_REQUESTED( self):
                    return iIOs().idinps_rop().idinp_ROP_START().value() == 1

                def exitcondition_ROPSTOP_IS_REQUESTED( self):
                    return iIOs().idinps_rop().idinp_ROP_STOP().value() == 1

                def exitcondition_ROPPAUSE_IS_REQUESTED( self):
                    return iIOs().idinps_rop().idinp_ROP_PAUSE().value() == 1

                def exitcondition_ROPRESUME_IS_REQUESTED( self):
                    return iIOs().idinps_rop().idinp_ROP_RESUME().value() == 1

                def robot( self):
                    return self.common().robot()
                def world( self):
                    return self.common().world()


            class ROPSMAvoiding(_SMState):

                def execute( self):
                    alpha = iIOs().iainps_rop().iainp_ROP_ALPHA().value()
                    if degrees( alpha) > 90:
                                                    # Obstacle approaches from the rhs
                        self.robot().uck_100( v_100=75, omega_100=-50)

                    elif degrees( alpha) < 90:
                                                    # Obstacle approaches from the lhs
                        self.robot().uck_100( v_100=75, omega_100=+50)

                    self.robot().execute()
                    return

                def exitcondition_ROPSTOP_IS_REQUESTED( self):
                    b = iIOs().idinps_rop().idinp_ROP_STOP().value() != 0
                    if b:
                        this_name = ThisName( self)
                        UsrEventLog().log_warning( this_name, this_name)

                    return b

                def exitcondition_NO_OBSTACLE_DETECTED( self):
                    b = iIOs().idinps_rop().idinp_ROP_ALPHA_DEVIATES().value() == 0
                    return b

                def exitcondition_VERY_CLOSE_OBSTACLE_DETECTED( self):
                    b = False
                    if iIOs().idinps_rop().idinp_ROP_ALPHA_DEVIATES().value() != 0:
                        if not 85 <= degrees( iIOs().iainps_rop().iainp_ROP_ALPHA().value()) <= 105:
                            b = True

                    return b


            class ROPSMEscaping(_SMState):

                def execute( self):
                    alpha = iIOs().iainps_rop().iainp_ROP_ALPHA().value()
                    if degrees( alpha) > 90:
                                                    # Obstacle approaches from the rhs
                        self.robot().uck_100( v_100=25, omega_100=-100)

                    elif degrees( alpha) < 90:
                                                    # Obstacle approaches from the lhs
                        self.robot().uck_100( v_100=25, omega_100=+100)

                    self.robot().execute()
                    return


            class ROPSMIdle(_SMState):

                def close( self):
                    super().close()

                    iIOs().idouts_rop().idout_ROP_IS_ENABLED().value( 1)
                                                    # Wir zeigen an, dass wir enabled-t sind.
                    return

                def exitcondition_IS_READY( self):
                    '''We may change over to READY.

                    Conditions:

                    -   Batteries ready
                    -   GPS ready
                    '''
                    is_batteries_ready = True
                    is_navi_ready = len( self.world().robots()) and
self.common().robot().sensors().navis( 0).status().is_ok()
                    is_rop_enabled = iIOs().idinps_rop().idinp_ROP_ENABLE().value() == 1
                                                    # PLC enable-t uns
                    if False not in (is_navi_ready, is_rop_enabled, is_batteries_ready):
                        UsrEventLog().log_info( "Batteries are ok and NavSys and ROP are ready. ", ThisName( self))

                    return is_navi_ready and is_rop_enabled

                def open( self, *args):
                    super().open( *args)

                    #self.robot().uck_100( v_100=0, omega_100=0)
                    #self.robot().execute()
                    # ##### Das Environment ist noch nicht aufgesetzt. 2DO: Wie können wir das sicherstellen?
                    return

            class ROPSMGoaling(_SMState):

                def close( self):
                    super().close()
                def exitcondition_NAVI_IS_NOT_READY( self):
                    b = iIOs().idinps_rop().idinp_ROP_IS_NAVIDATA_OK().value() == 0
                                                    # PLC sends us a 1, if it is receiving
                                                    #   valid data, and a 0 otherwise
                    if b:
                        this_name = ThisName( self)
                        UsrEventLog().log_warning( this_name, this_name)

                    return b


            class ROPSMLoading(_SMState):

                def exitcondition_MAP_IS_LOADED( self):
                    return True # 2DO


            class ROPSMPausing(_SMState):

                def close( self):
                    super().close()

                    if self.__is_stop_requested:
                        self.robot().stop()
                        return

                    if self.__is_resume_requested:
                        self.robot().stop()
                        return

                    if self.__is_diable_requested:
                        self.robot().stop()
                        iIOs().idouts_rop().idout_ROP_IS_ENABLED().value( 0)
                                                        # Wir zeigen an, dass wir disablet sind.
                        return

                    return

                def execute( self):
                    self.robot().uck_100( v_100=0, omega_100=0)
                    self.robot().execute()
                    return

                def exitcondition_STOP_IS_REQUESTED( self):
                    self.__is_stop_requested = iIOs().idinps_rop().idinp_ROP_STOP().value() # S T O P -Button
                    return self.__is_stop_requested

                def exitcondition_RESUME_IS_REQUESTED( self):
                    self.__is_resume_requested = iIOs().idinps_rop().idinp_ROP_RESUME().value() # R E S U M E -Button
                    return self.__is_resume_requested

                def exitcondition_DISABLE_IS_REQUESTED( self):
                    self.__is_diable_requested = iIOs().idinps_rop().idinp_ROP_ENABLE().value() == 0    # D I S A B L
E -Button
                                                    # PLC disablet uns
                    return self.__is_diable_requested

                def open( self, *args):
                    super().open( *args)

                    self.__is_diable_requested = False
                    self.__is_resume_requested = False
                    self.__is_stop_requested = False
                    return


            class ROPSMReady(_SMState):

                def close( self):
                    super().close()

                    if self.__is_ropstart_requested:
                        self.robot().start()
                        return

                    if self.__is_ropdisable_requested:
                        iIOs().idouts_rop().idout_ROP_IS_ENABLED().value( 0)
                                                        # Wir zeigen an, dass wir disablet sind.
                        return

                    if self.__is_ropenable_requested:
                        iIOs().idouts_rop().idout_ROP_IS_ENABLED().value( 1)
                                                        # Wir zeigen an, dass wir enabled-t sind.
                    return

                def execute( self):
                    self.robot().uck_100( v_100=0, omega_100=0)
                    self.robot().execute()
                                                    # Robot ausführen: Behaviours und Chassis.
                                                    #   Die Sensoren werden im SensorReader
                                                    #   gelesen.
                    return

                def exitcondition_ROPSTART_IS_REQUESTED( self):
                    self.__is_ropstart_requested = super().exitcondition_ROPSTART_IS_REQUESTED()
                    if self.__is_ropstart_requested:
                        this_name = ThisName( self)
                        UsrEventLog().log_info( this_name, this_name)

                    return self.__is_ropstart_requested

                def exitcondition_ROPDISABLE_IS_REQUESTED( self):
                    self.__is_ropdisable_requested = super().exitcondition_ROPDISABLE_IS_REQUESTED()
                    if self.__is_ropdisable_requested:
                        this_name = ThisName( self)
                        UsrEventLog().log_info( this_name, this_name)

                    return self.__is_ropdisable_requested

                def exitcondition_ROPENABLE_IS_REQUESTED( self):
                    self.__is_ropenable_requested = super().exitcondition_ROPENABLE_IS_REQUESTED()
                    if self.__is_ropenable_requested:
                        this_name = ThisName( self)
                        UsrEventLog().log_warning( this_name, this_name)

                    return self.__is_ropenable_requested

                def open( self, *args):
                    super().open( *args)

                    self.__is_ropdisable_requested = False
                    self.__is_ropenable_requested = False
                    self.__is_ropstart_requested = False
                    return


            class ROPSMWaitingForNavi(_SMState):

                def close( self):
                    super().close()

                    if self.__is_stop_requested:
                        self.robot().stop()
                        return

                    if self.__is_resume_requested:
                        self.robot().stop()
                        return

                    if self.__is_diable_requested:
                        self.robot().stop()
                        iIOs().idouts_rop().idout_ROP_IS_ENABLED().value( 0)
                                                        # Wir zeigen an, dass wir disablet sind.
                        return

                    return

                def execute( self):
                    self.robot().uck_100( v_100=0, omega_100=0)
                    self.robot().execute()
                    return

                def exitcondition_STOP_IS_REQUESTED( self):
                    self.__is_stop_requested = iIOs().idinps_rop().idinp_ROP_STOP().value() # S T O P -Button
                    return self.__is_stop_requested

                def exitcondition_NAVI_IS_READY( self):
                    b = iIOs().idinps_rop().idinp_ROP_IS_NAVIDATA_OK().value() == 1
                                                    # PLC sends us a 1, if it is receiving
                                                    #   valid data, and a 0 otherwise
                    if b:
                        this_name = ThisName( self)
                        UsrEventLog().log_warning( this_name, this_name)

                    return b

                def exitcondition_DISABLE_IS_REQUESTED( self):
                    self.__is_diable_requested = iIOs().idinps_rop().idinp_ROP_ENABLE().value() == 0    # D I S A B L
E -Button
                                                    # PLC disablet uns
                    return self.__is_diable_requested

                def open( self, *args):
                    super().open( *args)

                    self.__is_diable_requested = False
                    self.__is_resume_requested = False
                    self.__is_stop_requested = False
                    return


            class ROPSMNone(_SMState):

                def is_none( self):
                    False


            ############################################################################
            ### Attributes And Methods
            #
            def __init__( self):
                self.__sm = SM( self.table(), self.ROPSMIdle(), _Common())
                return
            def sm( self):
                return self.__sm

            def table( self):
                d = {\
                        self.ROPSMAvoiding(): \
                            {\
                                self.ROPSMAvoiding().exitcondition_NO_OBSTACLE_DETECTED: self.ROPSMGoaling(),
                                self.ROPSMAvoiding().exitcondition_VERY_CLOSE_OBSTACLE_DETECTED: self.ROPSMEscaping(),
                                self.ROPSMAvoiding().exitcondition_ROPDISABLE_IS_REQUESTED: self.ROPSMIdle(),
                                self.ROPSMAvoiding().exitcondition_ROPSTOP_IS_REQUESTED: self.ROPSMReady(),
                            },

                        self.ROPSMIdle(): \
                            {\
                                self.ROPSMIdle().exitcondition_IS_READY: self.ROPSMReady(),
                            },

                        self.ROPSMReady(): \
                            {\
                                self.ROPSMReady().exitcondition_ROPSTART_IS_REQUESTED: self.ROPSMLoading(),
                                self.ROPSMReady().exitcondition_ROPDISABLE_IS_REQUESTED: self.ROPSMIdle(),
                            },

                        self.ROPSMLoading(): \
                            {\
                                self.ROPSMLoading().exitcondition_MAP_IS_LOADED: self.ROPSMGoaling(),
                            },

                        self.ROPSMGoaling(): \
                            {\
                                self.ROPSMGoaling().exitcondition_ROPSTOP_IS_REQUESTED: self.ROPSMReady(),
                                self.ROPSMGoaling().exitcondition_ROPDISABLE_IS_REQUESTED: self.ROPSMIdle(),
                                self.ROPSMGoaling().exitcondition_ROPPAUSE_IS_REQUESTED: self.ROPSMPausing(),

                                self.ROPSMGoaling().exitcondition_NAVI_IS_NOT_READY: self.ROPSMWaitingForNavi(),
                            },

                        self.ROPSMPausing(): \
                            {\
                                self.ROPSMPausing().exitcondition_ROPSTOP_IS_REQUESTED: self.ROPSMReady(),
                                self.ROPSMPausing().exitcondition_ROPDISABLE_IS_REQUESTED: self.ROPSMIdle(),
                                self.ROPSMPausing().exitcondition_ROPRESUME_IS_REQUESTED: self.ROPSMGoaling(),
                            },

                        self.ROPSMWaitingForNavi(): \
                            {\
                                self.ROPSMWaitingForNavi().exitcondition_ROPSTOP_IS_REQUESTED: self.ROPSMReady(),
                                self.ROPSMWaitingForNavi().exitcondition_ROPDISABLE_IS_REQUESTED: self.ROPSMIdle(),

                                self.ROPSMWaitingForNavi().exitcondition_NAVI_IS_READY: self.ROPSMGoaling(),
                            },

                        self.ROPSMNone(): \
                            {\
                                self.ROPSMNone().is_none: self.ROPSMNone(),
                            },

                    }
                return d

    """

    def __init__( self, sms_table, sms_initial, sms_common_data):
        self.__sms_table = sms_table
        self.__sms_current = sms_initial
        self.__sms_common_data = sms_common_data

        self.__sms_current.open( self.__sms_common_data)

        self.__fv_smstate_name = flex.VariableDeMo( id=-1, value="???", label="SM State")
        self.__fv_smstate_number = flex.VariableDeMo( id=-1, value=-1, label="SM State")

        self.__is_finished = False
        return

    def common( self):
        return self.__sms_common_data

    def execute( self):
        if self.is_finished():
            UsrEventLog.log_error( "Cannot execute a state machine that has finished already!", ThisName( self))
            return

        self.__sms_current.execute()
        self.__fv_smstate_name.value( self.__sms_current.name())
        self.__fv_smstate_number.value( self.__sms_current.value())
        try:
            for exitconditionmethod, sms_next in self.__sms_table[ self.__sms_current]:
                if exitconditionmethod():
                    self.__sms_current.close()
                                                    # Close this state
                    self.__sms_current = sms_next
                                                    # Get the next state and set
                                                    #   it as the (new) current one
                    if self.__sms_current is None:
                        self.__is_finished = True
                        break

                    self.__sms_current.open( self.__sms_common_data)
                                                    # Open the new current state
                    break

        except KeyError as e:
            UsrEventLog().log_error( e + ". You forgot to enter this state in your state table!", ThisName( self))

        return self

    def fv_smstatename_current( self):
        return self.__fv_smstate_name
    fv_smstate_name = fv_smstatename_current # DEPRECATED

    def fv_smstatenumber_current( self):
        return self.__fv_smstate_number

    def is_finished( self):
        return self.__is_finished

    def smstate_current( self):
        return self.__sms_current


class SMState(metaclass=Singleton):

    def __init__( self):
        self._tau4p_on_close = PublisherChannel.Synch( self)
                                        # Called before closing the state
        self._tau4p_on_execute = PublisherChannel.Synch( self)
                                        # Called before executing the state
        self._tau4p_on_opened = PublisherChannel.Synch( self)
                                        # Called after opening the state
        self.__common = None
        self.__is_open = False
        return

    def close( self):
        """Close the state.

        May be overridden, but doesn't need to be.
        """
        assert self.__is_open
        self._tau4p_on_close()
        self.__is_open = False
        return

    def common( self):
        return self.__common

    @abc.abstractmethod
    def execute( self):
        self._tau4p_on_execute()
        assert self.__is_open

    def name( self):
        return self.__class__.__name__

    def open( self, common):
        """Open the state.

        May be overridden, but doesn't need to be.

        In case, the overriding method needs to call this class' method!
        """
        self.__common = common
        self.__is_open = True
        self._tau4p_on_opened()
        return self

    @abc.abstractmethod
    def value( self):
        raise NotImplementedError()
```

6 emlid

6.1 reach

7 mathe – Mathematix Enhanced

7.1 geometry (.py, _py.py, _cy.pyx)

7.1.1 class Circle2D

7.1.1.1 Interface

__init__(self, T, r)

is_intersecting(self, other)

x(self)

y(self)

7.1.1.2 Usage

2DO

7.1.1.3 Known Use

Projects

- FHVROSIM

7.1.2 class Line2D

7.1.3 class Point2D

7.1.3.1 Interface

FromOther(other, T3=T3D.FromEuler(), scale_x=1, scale_y=1)

__init__(self, x=0, y=0)

__eq__(self, other)

__getitem__(self, i)

__repr__(self)

__setitem__(self, i, v)

__lshift__(self, other)

clipped(self, y_limits)

x(self)

y(self)

7.1.3.2 Discussion

Should we derive from `V3D`?

7.1.3.3 Known Use

- CRUISER

7.1.4 class Polygon2D

7.1.4.1 Interface

__init__(self, points: [Point2D])

“””Ctor.

:param points:

Plain list of Point2D.

is_point_inside(self, point: Point2D)

is_circle_inside(self, circle: Circle2D)

intersection_points_circle(self, circle: Circle2D)

7.1.5 class Sphere(Circle2D)

7.1.5.1 Interface

z(self)

7.1.5.2 Discussion

`Circle2D` could have inherited from `Sphere`. In that case `z()` would have returned zero, always.

7.1.5.3 Usage

2DO

7.1.5.4 Known Use

–

8 pandora

file:///home/fgeiger/D.X/Projects/tau4/swr/py3/dox/tau4.data.pandora.odt

9 oop

Tools And Utilities For SoftWare ENGineering.

This modules implements some design patterns, which proved being useful in Python programs.

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of tau4.
#
#   tau4 is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   tau4 is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with tau4. If not, see <http://www.gnu.org/licenses/>.


from __future__ import division

import logging; _Logger = logging.getLogger()


import abc
import inspect

import tau4
import threading


def overrides( interface_class):
    """Decorator."""
    def overrider( method):
        assert( method.__name__ in dir( interface_class))
        return method

    return overrider


class _PublisherChannel:

    """Basisklasse für alle Arten von Channeln für Instanzen, die publishen wollen.

    Parameters:
        publisher:
            Objekt, das diese Klasse instanziert hat und publishen will.

    Usage:
        p = PublisherChannel( self)
        p()
                                        # Schickt die PublisherChannel-Instanz
                                        #   an alle Subscriber,
                                        #   die sich registriert haben.

    Usage:
        p = PublisherChannel( self)
        p( 42)
                                        # Schickt die PublisherChannel-Instanz und
                                        #   den Wert 42 an alle Subscriber,
                                        #   die sich registriert haben.

    History:
        2014-09-06:
            Created.

            Unterschied zu _Publisher: Nur Namensgebung. Obwohl: Es wäre
            denkbar, dass einmal nicht die PublisherChannel-Instanz an die Subscriber
            verschickt wird, sondern die Instanz des Publishers selbst.
    """

    __metaclass__ = abc.ABCMeta

    def __init__( self, publisher):
        self.__publisher = publisher

        self.__subscribers = []
        self.__lock = threading.Lock()
        return

    @abc.abstractmethod
    def __call__( self, *args, **kwargs):
        pass

    def __len__( self):
        return len( self.__subscribers)

    def client( self):
        """Same as ``parent()``: Returns hosting (= publishing) instance.
        """
        return self.__publisher

    def is_empty( self):
        """Prüft, ob Handler auszuführen sind.
        """
        return len( self.__subscribers) == 0

    @abc.abstractmethod
    def is_sync( self):
        """Prüft, ob es sich um einen a/synchronen Handler handelt.
        """
        pass

    def parent( self):
        """Returns hosting instance.
        """
        return self.__publisher

    def publisher( self):
        """Returns hosting instance, which may be considered being the actual publisher.
        """
        return self.__publisher

    def subscriber_count( self):
        return len( self.__subscribers)

    def subscriber_register( self, subscriber):
        """Neuen Subscriber hinzufügen.

        Parameters:
            subscriber:
                callable, der ausgeführt werden soll. Damit ist jedes Objekt ein
                Subscriber, das callable ist, d.s. Methode oder Objekte, die die
                Methode __call__() implementieren!

        Raises:
            ValueError wenn der Subscriber bereits bekannt ist.
        """
        with self.__lock:
            if self.__subscribers.count( subscriber):
                raise ValueError( "Subscriber already registered!")

            self.__subscribers.append( subscriber)

        return self

    def subscriber_is_registered( self, subscriber):
        """Subscriber schon registriert?
        """
        with self.__lock:
            return self.__subscribers.count( subscriber) > 0

    def subscriber_un_register( self, subscriber):
        """Subscriber entfernen.
        """
        with self.__lock:
            if not self.__subscribers.count( subscriber):
                raise ValueError( "Subscriber not registered!")

            self.__subscribers.remove( subscriber)

        return self

    def subscriber_un_register_all( self):
        """Alle Handler entfernen.
        """
        with self.__lock:
            self.__subscribers[:] = []

        return self

    def subscribers( self, copy=True):
        """Liefert (Kopie der) Subscribers.
        """
        if copy:
            with self.__lock:
                return self.__subscribers[ :]

        return self.__subscribers


class PublisherChannel:
    """Namespace.

    Usage:
        tau4pc = PublisherChannel.Synch( self)

    History:
        2014-09-06:
```

```
        Created.
"""

class Synch(_PublisherChannel):

    """Siehe Base Class ``_Publisher``.
    """

    def __init__( self, parent):
        _PublisherChannel.__init__( self, parent)

        self.__is_safe_mode = False
        return

    def __call__( self, *args, **kwargs):
        """Ausführen aller registrierter Subscribers.

        Usage:
            Variante ohne Parent:
                p = _Publisher( None)
                p( 42)

                Da hier ``None`` als ``parent`` übergeben worden ist, kann/muss der
                Subscriber folgendermaßen definiert werden:
                    def _tau4s_on_data_( self, value):
                        return

            Variante mit Parent:
                p = _Publisher( self)
                p( 42)

                Da hier das hostende Objekt als ``parent`` übergeben worden ist,
                kann/muss der Subscriber folgendermaßen definiert werden:
                    def _tau4s_on_data_( self, tau4pc, value):
                        '''Subscriber.

                        Parameters:
                            tau4pc:
                                PublisherChannel.

                            value:
                                Additional arg sent by publisher.

                        Note:
                            You may get at the publishing object by a call to
                            ``tau4pc.publisher()``.
                        '''
                        return

        """
        ss = self.subscribers( copy=self.is_safe_mode())
        for s in ss:
            s( self, *args, **kwargs)

        return self

    def __iadd__( self, subscriber):
        """'Syntactic sugar', führt einfach subscriber_register() aus.
        """
        assert callable( subscriber)
        self.subscriber_register( subscriber)
        return self

    def __isub__( self, subscriber):
        """'Syntactic sugar', führt einfach subscriber_un_register() aus.
        """
        assert callable( subscriber)
        self.subscriber_un_register( subscriber)
        return self

    def is_sync( self):
        """
        """
        return True

    def is_safe_mode( self, arg=None):
        """Zugriff auf Subscribers über Zugriffsregelung per Lock?
        """
        if arg is None:
            return self.__is_safe_mode

        self.__is_safe_mode = arg
        return self


class Async(_PublisherChannel):

    """Siehe Base Class ``_Publisher``.

    Note:
        D214:
            Work in progress: Kann noch nicht instanziert werden, weil die Implementierung
            der __call__-Methode noch fehlt.
    """

    def __init__( self, parent):
        _PublisherChannel.__init__( self, parent)
```

```
            self.__is_safe_mode = True
            return

        def __iadd__( self, subscriber):
            """'Syntactic sugar', führt einfach subscriber_register() aus.
            """
            assert callable( subscriber)
            self.subscriber_register( subscriber)
            return self

        def __isub__( self, subscriber):
            """'Syntactic sugar', führt einfach subscriber_un_register() aus.
            """
            assert callable( subscriber)
            self.subscriber_un_register( subscriber)
            return self

        def is_sync( self):
            """
            """
            return False

        def is_safe_mode( self, arg=None):
            """Zugriff auf Subscribers über Zugriffsregelung per Lock?
            """
            if arg is None:
                return self.__is_safe_mode

            self.__is_safe_mode = arg
            return self


class Singleton(type):

    """ Thread-safe Singleton, after http://timka.org/tech/2008/12/17/singleton-in-python/.

    We can make any existing class a singleton by simply adding the __metaclass__ attribute.
    The only Singleton instance is stored in the __instance__ class attribute.
    However, there's a problem here: Note that the __init__() method is called
    on every instantiation. This is normal behaviour of types in Python. When
    you instantiate a class, __new__() and __init__() are called internally.
    But we want the single instance to be created and initialized only once.
    The only(?) way to achieve this is metaclasses. In metaclass you can define
    what happens when you call its instances (which are also classes).

    Usage:
        \

        ::

            def test( self):
                 print

                 class MySingleton:
                     __metaclass__ = Singleton

                     def __init__( self, a, b):
                         self._a = a
                         self._b = b
                         return

                     def __eq__( self, other):
                         if not isinstance( other, MySingleton):
                             return False

                         return self._a == other._a and self._b == other._b

                     def __ne__( self, other):
                         return not self == other


                 s1 = MySingleton( 1, 2)
                 s2 = MySingleton( 3, 4)
                 self.assertTrue( s1 == s2)
                 self.assertTrue( s1 is s2)

                 return
    """

    def __new__( klass, name, bases, namespace):
        namespace.setdefault( '__lock__', threading.RLock())
                                        # Allocate lock, if not already allocated manually.
        namespace.setdefault( '__instance__', None)
                                        # Since we are already using __new__,
                                        #   we can also initialize the
                                        #   __instance__ attribute here.
        return super( Singleton, klass).__new__( klass, name, bases, namespace)

    #   Define the __call__ method in metaclass where __new__() and __init__()
    #   are called manually and only once.
    #
    def __call__( klass, *args, **kwargs):
        klass.__lock__.acquire()
        try:
                                        # __instance__ is now always initialized,
                                        #   so no need to use a default value.
            if klass.__instance__ is None:
                instance = klass.__new__( klass, *args, **kwargs)
                instance.__init__( *args, **kwargs)
                klass.__instance__ = instance
        finally:
            klass.__lock__.release()

        return klass.__instance__
```

10 ce

10.1

11 com

12 data

Dieses Package besteht aus folgenden Packages:

- varbls

12.1 .varbls

12.1.1 FlexValue

Deckt die Features

- Reading / writing

ab. Alles weitere ist durch Subclasses zu realisieren, damit Objekte dieser Klasse schnell bleiben.

Das „Flex“ im Namen bedeutet, dass der Typ, den das Objekt hält, vom Wert abhängig ist, der dem Ctor übergeben wird.

Eine Identifikation des Values ist immer möglich, indem man

```
tau4.Objects.add( v, u”your sophisticated name goes here”)
```

ausführt, wobei `v` z.B. per

```
v = FlexValue( 42)
```

erzeugt worden ist.

12.1.2 FlexVarbl

Hinzu kommen die Features

- Identification
- Timing
  - Created
  - Modified
- Publishing
  - on_modified
  - on_limit_violated

Clipping und Scaling sind Strategien, die übergeben werden können. Eine Strategie wird von der Basisklasse `ValueMangler` abgeleitet.

12.1.3 FlexQuant

Hinzu kommen die Features

- Name
- Dimension

12.1.4 ValueMangler

Jede Strategie muss die beiden Methoden `app2value()` und `value2app()` implementieren.

12.1.4.1 Clipper

Wenn man clippen möchte, dann funktioniert das so:

```
v = varbls.FlexVarbl( 0.0)
v.value_manglers().add( varbls.Clipper( -42, 42)))
```

Das Clipping erfolgt immer beim Schreiben. Für eine geclippte Varbl gilt also immer die Invariante

```
min <= v <= max
```

12.1.4.2 Scaler

Wenn die App in anderen Dimensionen rechnet als das System, dann kann man so vorgehen:

```
v = varbls.FlexVarbl( 0.0)
                                # Rechnet in mm
v.value_manglers().add( varbls.Scaler( app2value=1000)))
                                # Rechnet in m.
                                # Beim Schreiben des value wird mit 1000 mult., beim
                                # Lesen durch 1000 div.
```

13 io – I/Os

13.1 hal – Hardware Abstraction Layer

API of `io`. As this is package serves as an abstraction layer, the concrete i/os have to be introduced to `hal`.

13.1.1 Usage

13.2 i2c

13.3 arduino

13.4 labjack

14 iio – Internal I/Os

14.1 2DO

- Umbenennen in vio – Virtual I/Os
-

15 sensors

15.1 Class Diagram *tau4.sensors*

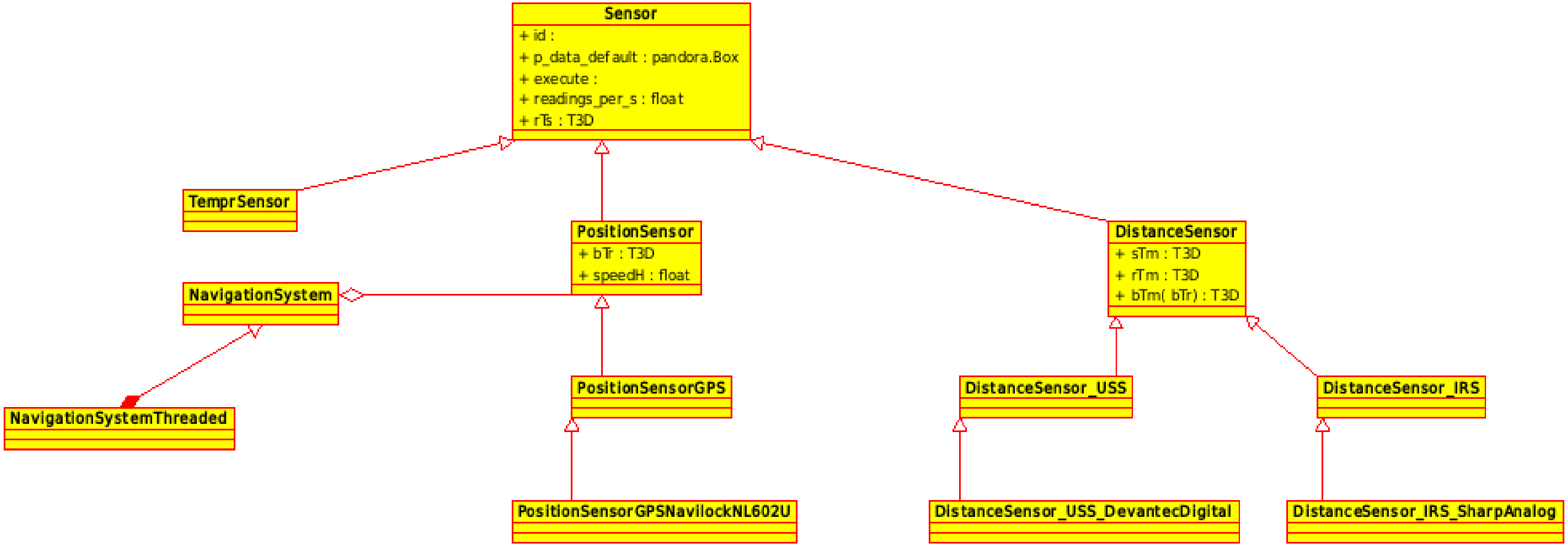

Fig. 15.1: Class Diagram *tau4.sensors*

16 Testsuites

16.1 automation

16.1.1 sm

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of tau4.
#
#   tau4 is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   tau4 is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with tau4. If not, see <http://www.gnu.org/licenses/>.


from __future__ import division

import logging; _Logger = logging.getLogger()
import socket

import tau4
from tau4 import ThisName
from tau4.datalogging import UsrEventLog
import time
import unittest

from tau4.automation.sm import SM, SMState


class _SMStates:

    class Idle(SMState):

        def __init__( self):
            super().__init__()
            return

        def execute( self):
            return

        def is_button_1_pressed( self):
            return True

        def is_button_2_pressed( self):
            return False


    class Finished(SMState):

        def __init__( self):
            super().__init__()
            return

        def execute( self):
            pass


    class Ready(SMState):

        def __init__( self):
            super().__init__()
            self.__time_created = time.time()
            return

        def execute( self):
            return

        def is_timeout( self):
            is_timeout = time.time() - self.__time_created > 1.5
            return is_timeout


class _TESTCASE__SM(unittest.TestCase):

    def test__simple( self):
        """
        """
        print()

        _SMSTable = {\
            _SMStates.Idle(): \
                {\
                    _SMStates.Idle().is_button_1_pressed: _SMStates.Ready(),
                    _SMStates.Idle().is_button_2_pressed: _SMStates.Finished()
                },

            _SMStates.Ready():
                { _SMStates.Ready().is_timeout: _SMStates.Finished()},

            _SMStates.Finished():
                { lambda: True: _SMStates.Finished()}
        }

        sm = SM( _SMSTable, _SMStates.Idle(), None)
        t = time.time()
        while time.time() - t < 2:
            print( "Current state = " + sm.smstate_current().__class__.__name__)
            sm.execute()
            time.sleep( 0.100)

        self.assertIs( sm.smstate_current(), _SMStates.Finished())

        return


_Testsuite = unittest.makeSuite( _TESTCASE__SM)


class _SMSStatesEmlidReach:

    class Idle(SMState):

        def execute( self):
            return

        def is_enabled( self):
            return True


    class Connecting(SMState):

        def __init__( self):
            super().__init__()

            self.__ip_addr, self.__ip_portnbr = "10.0.0.13", 1962
            self.__is_error = False
            self.__is_open = False
            return

        def execute( self):
            try:
                self.__socket = socket.socket( socket.AF_INET, socket.SOCK_STREAM)
                self.__socket.settimeout( 10)
                self.__socket.connect( (self.__ip_addr, self.__ip_portnbr))
                self.__is_open = True

            except socket.timeout as e:
                UsrEventLog().log_error( "Cannot connect to navi: '%s'!" % e, ThisName( self))

            except ConnectionRefusedError as e:
                UsrEventLog().log_error( "Cannot connect to navi: '%s'!" % e, ThisName( self))
                self.__is_error = True

            except OSError as e:
                UsrEventLog().log_error( "Cannot connect to navi: '%s'!" % e, ThisName( self))
                self.__is_error = True

            return self

        def is_connected( self):
            return self.__is_open

        def is_error( self):
            return self.__is_error


    class Connected(SMState):

        def execute( self):
            return

        def is_disconnected( self):
            return True


    class Error(SMState):

        def execute( self):
            return

        def is_acked( self):
            return True
```

```
class _TESTCASE__EMlidREach(unittest.TestCase):

    def test( self):
        """
        """
        print()

        _SMSTable = {\
            _SMSStatesEmlidReach.Idle():\
                { _SMSStatesEmlidReach.Idle().is_enabled: _SMSStatesEmlidReach.Connecting()},

            _SMSStatesEmlidReach.Connecting():\
                {\
                    _SMSStatesEmlidReach.Connecting().is_connected: _SMSStatesEmlidReach.Connected(),
                    _SMSStatesEmlidReach.Connecting().is_error: _SMSStatesEmlidReach.Error()
                },

            _SMSStatesEmlidReach.Connected():\
                { _SMSStatesEmlidReach.Connected().is_disconnected: _SMSStatesEmlidReach.Connecting()},

            _SMSStatesEmlidReach.Error():\
                { _SMSStatesEmlidReach.Error().is_acknd: _SMSStatesEmlidReach.Idle()},
        }

        sm = SM( _SMSTable, _SMSStatesEmlidReach.Idle(), None)
        t = time.time()
        while time.time() - t < 2:
            print( "Current state = " + sm.smstate_current().__class__.__name__)
            sm.execute()
            time.sleep( 0.100)

        return


_Testsuite.addTest( unittest.makeSuite( _TESTCASE__EMlidREach))


class _TESTCASE__(unittest.TestCase):

    def test( self):
        """
        """
        print()
        return


_Testsuite.addTest( unittest.makeSuite( _TESTCASE__))


def _lab_():
    return


def _Test_():
    unittest.TextTestRunner( verbosity=2).run( _Testsuite)


if __name__ == '__main__':
    _Test_()
    _lab_()
    input( u"Press any key to exit...")
```

16.2 sweng

17 Appendix:

18 Appendix: Document Index

tau4.data

file:///home/fgeiger/D.X/Projects/tau4/swr/py3/dox/_tau4data.odt

~~pandora~~

~~Formerly known as `tau4.data.flex`.~~

~~file:///home/fgeiger/D.X/Projects/pandora/swr/py3/dox/pandora.odt~~

tau4.data.pandora

file:///home/fgeiger/D.X/Projects/tau4/swr/py3/dox/tau4.data.pandora.odt

ios

Formerly known as `tau4.io` and `tau4.iio`.

file:///home/fgeiger/D.X/Projects/ios/swr/py3/dox/ios.odt

Index