```
-- +=======================================================+
-- | CSP Model of the overall dynamic behavior for the     |
-- | Uni-processor Simplex Architecture.                   |
-- |                                                       |
-- | Author:  Jose German Rivera                           |
-- |          Alejandro Andres Danylyszyn                  |
-- |                                                       |
-- | Created: 10/4/95  - Initial version: Upgrade Manager  |
-- |                     not included.                     |
-- +=======================================================+

-- +=======================================================+
-- | Channel Alphabets                                     |
-- +=======================================================+

-- Events received from the controlled plant:
FROMPLANT = {safe, unsafe}

-- Events sent to the controlled plant:
TOPLANT = {cntrlout}

-- Events generated by the controllers:
CNTRLEVT = {cntrlout, failstop, illegalout, responseTimeout}

-- Event sent to the controllers:
TOCNTRL = {kill}

-- +=======================================================+
-- | Channel Declarations                                  |
-- +=======================================================+

pragma channel fromPlant, fromIO: FROMPLANT
pragma channel toPlant, toIO: TOPLANT
pragma channel fromComplex, fromBaseline, fromSafety: CNTRLEVT
pragma channel toComplex, toBaseline, toSafety: TOCNTRL

-- +=======================================================+
-- | Process Definitions                                   |
-- +=======================================================+

--
-- Simplex architecture:
--

SIMPLEX = (DECISION
           [| {| fromIO, toIO |} |]
           PHYSICALIO)
          [| {| fromComplex, toComplex, fromBaseline, toBaseline,
                fromSafety, toSafety |} |]
          (COMPLEX ||| BASELINE ||| SAFETY)

--
-- Decision Component:
--
DECISION = COMPLEXLOOP

COMPLEXLOOP = fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
              toComplex!kill -> TEMPSAFETYLOOP
              []
              fromIO.safe -> (fromComplex.cntrlout -> toIO!cntrlout ->
                              COMPLEXLOOP
                              []
                              ([] x: {illegalout, responseTimeout} @
                                  fromComplex.x -> toComplex!kill ->
                                  BASELINESAFE)
                              []
                              fromComplex.failstop -> BASELINESAFE)

BASELINESAFE = fromBaseline.cntrlout -> toIO!cntrlout -> BASELINELOOP
               []
               fromBaseline.failstop -> fromSafety.cntrlout ->
               toIO!cntrlout -> SAFETYLOOP

BASELINELOOP = fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
               toBaseline!kill -> SAFETYLOOP
               []
               fromIO.safe -> BASELINESAFE

TEMPSAFETYLOOP = fromIO.safe -> BASELINESAFE
                 []
                 fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
                 TEMPSAFETYLOOP

SAFETYLOOP = fromIO?x -> fromSafety.cntrlout -> toIO!cntrlout -> SAFETYLOOP

--
--  Physical I/O component:
--
PHYSICALIO = INPUT ||| OUTPUT

-- Take input from Plant:
INPUT = fromPlant?status -> fromIO!status -> INPUT

-- Send output to Plant:
OUTPUT =  toIO?command -> toPlant!command -> OUTPUT

--
-- Complex Component:
--
COMPLEX = ((|~| x: {cntrlout, illegalout, responseTimeout} @
                fromComplex!x -> COMPLEX)
           []
           toComplex.kill -> SKIP)
          |~|
          (fromComplex!failstop -> SKIP
           []
           toComplex.kill -> SKIP)

--
-- Baseline Component:
--
BASELINE = (fromBaseline!cntrlout -> BASELINE
            []
            toBaseline.kill -> SKIP)
           |~|
           (fromBaseline!failstop -> SKIP
            []
            toBaseline.kill -> SKIP)

--
-- Safety Component:
--
SAFETY =  fromSafety!cntrlout -> SAFETY


-- +=======================================================+
-- | Properties to verify                                  |
-- +=======================================================+

-- Auxiliary definitions:
```

```
SIGMA = {| fromPlant, toPlant, fromIO, toIO,
           fromComplex, fromBaseline, fromSafety,
           toComplex, toBaseline, toSafety |}

pragma channel e


-- Property 1. Deadlock-free.
--      |-  P1 is failure-divergence-refined by P1SIMPLEX
-- where:

P1 = e -> P1

P1SIMPLEX = identify (SIGMA, e, SIMPLEX)


-- Property 2. Control commands are sent to the plant infinitely often. That is,
-- it is never the case that from a given instant commands are not sent
-- to the plant anymore.
--      |- P2 is failure-divergence-refined by P2SIMPLEX
-- where:

P2 = |~| x: TOPLANT @ toIO!x -> P2

P2SIMPLEX = SIMPLEX \ diff (SIGMA, {| toIO |})


-- Property 3. It is never the case that there are two consecutive readings from
-- the plant without an output command between them. In other words, after
-- reading the status of the plant, and before performing the next reading,
-- a command has to be sent to plant. (This could be a way of detecting that
-- a deadline was missed)
--      |-  P3 is trace-refined by P3SIMPLEX
-- where:

P3 = fromIO?x -> toIO!cntrlout -> P3

P3SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, toIO |})


-- Property 4. After unsafe condition the plant is immediately controlled by
-- safety, and as long as it is unsafe it remains in control by safety:
--      |-  P4 is trace-refined by P4SIMPLEX
-- where:

P4 =  fromIO.unsafe -> fromSafety.cntrlout -> toIO.cntrlout -> P4
      |~|
      fromIO.safe -> (|~| c: { fromComplex, fromBaseline, fromSafety } @
                        c.cntrlout -> toIO.cntrlout -> P4)

P4SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, toIO, fromSafety.cntrlout,
                                      fromComplex.cntrlout,
                                      fromBaseline.cntrlout |})


-- Property 5. Whenever the plant is in safe state and is being controlled
-- by the complex controller, if the complex controller does not
-- produce an output on time (i.e. it misses its deadline or falls into an
-- infinite loop) or if the output is illegal, the control of the device is
-- passed to the baseline controller (or to the safety controller, in case
-- the baseline controller fails).
--      |- P5 is trace-refined by P5SIMPLEX
-- where:

P5 = fromIO.safe ->
     ((|~| x: {illegalout, responseTimeout, failstop} @
          fromComplex.x -> (|~| c: {fromBaseline, fromSafety} @
                              c.cntrlout -> P5))
      |~|
      (|~| c: {fromComplex, fromBaseline, fromSafety} @ c.cntrlout -> P5)
     )
     |~|
     fromIO.unsafe -> fromSafety.cntrlout -> P5

P5SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, fromComplex,
                                      fromBaseline.cntrlout,
                                      fromSafety.cntrlout |})
```

```
-- +=======================================================+
-- | CSP Model of the overall dynamic behavior for the     |
-- | Uni-processor Simplex Architecture.                   |
-- |                                                       |
-- | Author:  Jose German Rivera                           |
-- |          Alejandro Andres Danylyszyn                  |
-- |                                                       |
-- | Created: 10/4/95  - Initial version: Upgrade Manager  |
-- |                     not included.                     |
-- | Updated: 11/12/95 - Included Upgrade Manager. It has a|
-- |                     problem: properties 1-2 are not   |
-- |                     satisfied due to the fact that some|
-- |                     special cases of starting Complex |
-- |                     or killing Baseline/Complex were  |
-- |                     not considered. Property 6 added. |
-- +=======================================================+

-- +=======================================================+
-- | Channel Alphabets                                     |
-- +=======================================================+

-- Events received from the controlled plant:
FROMPLANT = {safe, unsafe}

-- Events sent to the controlled plant:
TOPLANT = {cntrlout}

-- Events generated by the controllers:
CNTRLEVT = {cntrlout, illegalout, responseTimeout}

-- Events sent from the Upgrade Manager to the controllers:
UMtoCTRL = {start, enableOutput, kill}

-- Events sent from the controllers to the Upgrade Manager:
CTRLtoUM = {initDone, dead}

-- Events sent from Decision to the Upgrade Manager:
DtoUM = {killBaseline, killComplex}

-- Events sent from the Upgrade Manager to Decision:
UMtoD = {baselineRunning, complexRunning, userKillBaseline,
         userKillComplex}

-- Events Received from the user:
FROMUSER = {startBaseline, startComplex, killBaseline,
            killComplex}

-- +=======================================================+
-- | Channel Declarations                                  |
-- +=======================================================+

pragma channel fromPlant, fromIO: FROMPLANT
pragma channel toPlant, toIO: TOPLANT
pragma channel fromComplex, fromBaseline, fromSafety: CNTRLEVT
pragma channel UMtoComplex, UMtoBaseline: UMtoCTRL
pragma channel ComplexToUM, BaselineToUM: CTRLtoUM
pragma channel DecisionToUM: DtoUM
pragma channel UMtoDecision: UMtoD
pragma channel fromUser: FROMUSER

-- +=======================================================+
-- | Independent events observed/produced by the Upgrade Manager |
-- +=======================================================+

pragma channel convergenceDetected
pragma channel convergenceTimeout
pragma channel raiseBaselinePrio
pragma channel raiseComplexPrio
pragma channel lowerBaselinePrio
pragma channel lowerComplexPrio

-- +=======================================================+
-- | Process Definitions                                   |
-- +=======================================================+

--
-- Simplex architecture:
--
SIMPLEX =
    UPGRADEMGR
    [| {| UMtoComplex, ComplexToUM, UMtoBaseline, BaselineToUM,
          DecisionToUM, UMtoDecision |} |]
    ((DECISION
      [| {| fromComplex, fromBaseline, fromSafety |} |]
      (COMPLEX ||| BASELINE ||| SAFETY))
     [| {| fromIO, toIO |} |]
     PHYSICALIO)

--
-- Upgrade Manager Component:
--
UPGRADEMGR = WILLINGTOSTARTBASELINE

WILLINGTOSTARTBASELINE =
   fromUser.startBaseline -> UMtoBaseline!start ->
   BaselineToUM.initDone -> raiseBaselinePrio ->
   (convergenceDetected -> UMtoBaseline!enableOutput ->
    UMtoDecision!baselineRunning -> WILLINGTOSTARTCOMPLEX
    |~|
    convergenceTimeout -> KILLBASELINE)

WILLINGTOSTARTCOMPLEX =
    fromUser.startComplex -> STARTCOMPLEX
    []
    DecisionToUM.killBaseline -> KILLBASELINE
    []
    fromUser.killBaseline -> UMtoDecision!userKillBaseline -> KILLBASELINE

KILLBASELINE =
    lowerBaselinePrio -> UMtoBaseline!kill ->
    BaselineToUM.dead -> WILLINGTOSTARTBASELINE

STARTCOMPLEX =
    UMtoComplex!start -> ComplexToUM.initDone -> raiseComplexPrio ->
    (convergenceTimeout -> KILLCOMPLEX
     |~|
     convergenceDetected -> UMtoComplex!enableOutput ->
     UMtoDecision!complexRunning ->
     (fromUser.killComplex -> UMtoDecision!userKillComplex -> KILLCOMPLEX
      []
      DecisionToUM.killComplex -> KILLCOMPLEX))

KILLCOMPLEX =
    lowerComplexPrio -> UMtoComplex!kill -> ComplexToUM.dead ->
    WILLINGTOSTARTCOMPLEX

--
-- Decision Component:
--
DECISION = SAFETYLOOP
```

```
SAFETYLOOP =
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    SAFETYLOOP
    []
    fromIO.safe -> (UMtoDecision.baselineRunning -> BASELINESAFE
                    []
                    fromSafety.cntrlout -> toIO!cntrlout ->
                    SAFETYLOOP)

BASELINELOOP =
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    DecisionToUM!killBaseline -> SAFETYLOOP
    []
    fromIO.safe -> (UMtoDecision.complexRunning -> COMPLEXSAFE
                    []
                    BASELINESAFE)
    []
    UMtoDecision.userKillBaseline -> SAFETYLOOP

BASELINESAFE =
    fromBaseline.cntrlout -> toIO!cntrlout -> BASELINELOOP
    []
    ([] x: {illegalout, responseTimeout} @
        fromBaseline.x -> DecisionToUM!killBaseline ->
        fromSafety.cntrlout -> toIO!cntrlout -> SAFETYLOOP)

COMPLEXLOOP =
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    DecisionToUM!killComplex -> TEMPSAFETYLOOP
    []
    fromIO.safe -> COMPLEXSAFE
    []
    UMtoDecision.userKillComplex -> BASELINELOOP

COMPLEXSAFE =
    fromComplex.cntrlout -> toIO!cntrlout -> COMPLEXLOOP
    []
    ([] x: {illegalout, responseTimeout} @
        fromComplex.x -> DecisionToUM!killComplex ->
        BASELINESAFE)

TEMPSAFETYLOOP =
    fromIO.safe -> BASELINESAFE
    []
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    TEMPSAFETYLOOP

--
-- Safety Component:
--
SAFETY =  fromSafety!cntrlout -> SAFETY

--
-- Baseline Component:
--
BASELINE =
    UMtoBaseline.start -> BaselineToUM!initDone ->
    (UMtoBaseline.enableOutput -> BASELINERUNNING
     []
     UMtoBaseline.kill -> BaselineToUM!dead -> BASELINE)

BASELINERUNNING =
    (|~| x: {cntrlout, illegalout, responseTimeout} @
         fromBaseline!x -> BASELINERUNNING)
    []
    UMtoBaseline.kill -> BaselineToUM!dead -> BASELINE

--
-- Complex Component:
--
COMPLEX =
    UMtoComplex.start -> ComplexToUM!initDone ->
    (UMtoComplex.enableOutput -> COMPLEXRUNNING
     []
     UMtoComplex.kill -> ComplexToUM!dead -> COMPLEX)

COMPLEXRUNNING =
    (|~| x: {cntrlout, illegalout, responseTimeout} @
         fromComplex!x -> COMPLEXRUNNING)
    []
    UMtoComplex.kill -> ComplexToUM!dead -> COMPLEX

--
--  Physical I/O component:
--
PHYSICALIO = INPUT ||| OUTPUT

-- Take input from the Plant:
INPUT = fromPlant?status -> fromIO!status -> INPUT

-- Send output to the Plant:
OUTPUT =  toIO?command -> toPlant!command -> OUTPUT


-- +=======================================================+
-- | Properties to verify                                  |
-- +=======================================================+

-- Auxiliary definitions:
SIGMA = {| fromUser, fromPlant, toPlant, fromIO, toIO,
           fromComplex, fromBaseline, fromSafety,
           UMtoComplex, UMtoBaseline, ComplexToUM,
           BaselineToUM, DecisionToUM, UMtoDecision,
           convergenceDetected,
           convergenceTimeout,
           raiseBaselinePrio,
           raiseComplexPrio,
           lowerBaselinePrio,
           lowerComplexPrio |}

pragma channel e


-- Property 1. Deadlock-free.
--      |-  P1 is failure-divergence-refined by P1SIMPLEX
-- where:

P1 = e -> P1

P1SIMPLEX = identify (SIGMA, e, SIMPLEX)


-- Property 2. Control commands are sent to the plant infinitely often. That
-- is, it is never the case that from a given instant commands are not sent
-- to the plant anymore.
--      |- P2 is failure-divergence-refined by P2SIMPLEX
-- where:

P2 = |~| x: TOPLANT @ toIO!x -> P2
```

```
P2SIMPLEX = SIMPLEX \ diff (SIGMA, {| toIO |})


-- Property 3. It is never the case that there are two consecutive readings
-- from the plant without an output command between them. In other words,
-- after reading the status of the plant, and before performing the next
-- reading, a command has to be sent to the plant. (This could be a way of
-- detecting that a deadline was missed)
--      |-  P3 is trace-refined by P3SIMPLEX
-- where:

P3 = fromIO?x -> toIO!cntrlout -> P3

P3SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, toIO |})


-- Property 4. After unsafe condition the plant is immediately controlled by
-- safety, and as long as it is unsafe it remains in control by safety:
--      |-  P4 is trace-refined by P4SIMPLEX
-- where:

P4 =  fromIO.unsafe -> fromSafety.cntrlout -> toIO.cntrlout -> P4
      |~|
      fromIO.safe -> (|~| c: { fromComplex, fromBaseline, fromSafety } @
                        c.cntrlout -> toIO.cntrlout -> P4)

P4SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, toIO, fromSafety.cntrlout,
                                   fromComplex.cntrlout,
                                   fromBaseline.cntrlout |})


-- Property 5. Whenever the plant is in safe state and is being controlled
-- by the complex controller, if the complex controller does not
-- produce an output on time (i.e. it misses its deadline or falls into an
-- infinite loop) or if the output is illegal, the control of the device is
-- passed to the baseline controller (or to the safety controller, in case
-- the baseline controller fails).
--      |- P5 is trace-refined by P5SIMPLEX
-- where:

P5 = fromIO.safe ->
     (((|~| x: {illegalout, responseTimeout} @
          fromComplex.x -> (|~| c: {fromBaseline, fromSafety} @
                            c.cntrlout -> P5))
      |~|
      (|~| c: {fromComplex, fromBaseline, fromSafety} @ c.cntrlout -> P5)
     )
     |~|
     fromIO.unsafe -> fromSafety.cntrlout -> P5

P5SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, fromComplex,
                                   fromBaseline.cntrlout,
                                   fromSafety.cntrlout |})


-- Property 6. Whenever Complex is started, Baseline has to be running, and
-- it never happens that there is more than one Baseline or more than one
-- Complex running.
--      |- P6 is trace-refined by P6SIMPLEX
-- where:

P6 = UMtoBaseline.start -> P6AUX
P6AUX = UMtoComplex.start -> UMtoComplex.kill -> P6AUX
        |~|
        UMtoBaseline.kill -> P6

P6SIMPLEX = SIMPLEX \ diff (SIGMA, { UMtoBaseline.start, UMtoBaseline.kill,
                                     UMtoComplex.start, UMtoComplex.kill })
```

```
-- +=====================================================+
-- | CSP Model of the overall dynamic behavior for the   |
-- | Uni-processor Simplex Architecture.                 |
-- |                                                     |
-- | Author:  Jose German Rivera                         |
-- |          Alejandro Andres Danylyszyn                |
-- |                                                     |
-- | Created: 10/4/95  - Initial version: Upgrade Manager|
-- |                     not included.                   |
-- | Updated: 11/12/95 - Included Upgrade Manager. It has a |
-- |                     problem: properties 1-2 are not |
-- |                     satisfied due to the fact that some|
-- |                     special cases of starting Complex  |
-- |                     or killing Baseline/Complex were   |
-- |                     not considered. Property 6 added.  |
-- | Updated: 11/24/95 - Property 1 now is satisfied, but   |
-- |                     property 2 is not completely       |
-- |                     satisfied (check2 succeeds but not |
-- |                     check1); reason being a divergence |
-- |                     caused by lack of fairness in FDR. |
-- +=====================================================+

-- +=====================================================+
-- | Channel Alphabets                                   |
-- +=====================================================+

-- Events received from the controlled plant:
FROMPLANT = {safe, unsafe}

-- Events sent to the controlled plant:
TOPLANT = {cntrlout}

-- Events generated by the controllers:
CNTRLEVT = {cntrlout, illegalout, responseTimeout}

-- Events sent from the Upgrade Manager to the controllers:
UMtoCTRL = {start, enableOutput, kill}

-- Events sent from the controllers to the Upgrade Manager:
CTRLtoUM = {initDone, dead}

-- Events sent from Decision to the Upgrade Manager:
DtoUM = {killBaseline, killComplex}

-- Events sent from the Upgrade Manager to Decision:
UMtoD = {baselineRunning, complexRunning, userKillBaseline,
         userKillComplex}

-- Events Received from the user:
FROMUSER = {startBaseline, startComplex, killBaseline,
            killComplex}

-- +=====================================================+
-- | Channel Declarations                                |
-- +=====================================================+

pragma channel fromPlant, fromIO: FROMPLANT
pragma channel toPlant, toIO: TOPLANT
pragma channel fromComplex, fromBaseline, fromSafety: CNTRLEVT
pragma channel UMtoComplex, UMtoBaseline: UMtoCTRL
pragma channel ComplexToUM, BaselineToUM: CTRLtoUM
pragma channel DecisionToUM: DtoUM
pragma channel UMtoDecision: UMtoD
pragma channel fromUser: FROMUSER
```

```
-- +=============================================================+
-- | Independent events observed/produced by the Upgrade Manager |
-- +=============================================================+

pragma channel convergenceDetected
pragma channel convergenceTimeout
pragma channel raiseBaselinePrio
pragma channel raiseComplexPrio
pragma channel lowerBaselinePrio
pragma channel lowerComplexPrio

-- +=====================================================+
-- | Process Definitions                                 |
-- +=====================================================+

--
-- Simplex architecture:
--
SIMPLEX =
    UPGRADEMGR
    [| {| UMtoComplex, ComplexToUM, UMtoBaseline, BaselineToUM,
          DecisionToUM, UMtoDecision |} |]
    ((DECISION
      [| {| fromComplex, fromBaseline, fromSafety |} |]
      (COMPLEX ||| BASELINE ||| SAFETY))
     [| {| fromIO, toIO |} |]
     PHYSICALIO)

--
-- Upgrade Manager Component:
--
UPGRADEMGR = WILLINGTOSTARTBASELINE

WILLINGTOSTARTBASELINE =
   fromUser.startBaseline -> UMtoBaseline!start ->
   BaselineToUM.initDone -> raiseBaselinePrio ->
   (convergenceDetected -> UMtoBaseline!enableOutput ->
    UMtoDecision!baselineRunning -> WILLINGTOSTARTCOMPLEX
    |~|
    convergenceTimeout -> KILLBASELINE)

WILLINGTOSTARTCOMPLEX =
    fromUser.startComplex -> STARTCOMPLEX
    []
    DecisionToUM.killBaseline -> KILLBASELINE
    []
    fromUser.killBaseline -> (UMtoDecision!userKillBaseline -> KILLBASELINE
                              []
                              DecisionToUM.killBaseline -> KILLBASELINE)

KILLBASELINE =
    lowerBaselinePrio -> UMtoBaseline!kill ->
    BaselineToUM.dead -> WILLINGTOSTARTBASELINE

STARTCOMPLEX =
    UMtoComplex!start -> ComplexToUM.initDone -> raiseComplexPrio ->
    (convergenceTimeout -> KILLCOMPLEX
     |~|
     convergenceDetected -> UMtoComplex!enableOutput ->
     (UMtoDecision!complexRunning ->
        (fromUser.killComplex ->
             (UMtoDecision!userKillComplex -> KILLCOMPLEX
              []
              DecisionToUM.killComplex -> KILLCOMPLEX)
         [])
```

```
          DecisionToUM.killComplex -> KILLCOMPLEX)
      []
      DecisionToUM.killBaseline -> lowerComplexPrio ->
      UMtoComplex!kill -> ComplexToUM.dead -> KILLBASELINE))

KILLCOMPLEX =
    lowerComplexPrio -> UMtoComplex!kill -> ComplexToUM.dead ->
    WILLINGTOSTARTCOMPLEX

--
-- Decision Component:
--
DECISION = SAFETYLOOP

SAFETYLOOP =
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    SAFETYLOOP
    []
    fromIO.safe -> (UMtoDecision.baselineRunning -> BASELINESAFE
                    []
                    fromSafety.cntrlout -> toIO!cntrlout ->
                    SAFETYLOOP)

BASELINELOOP =
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    DecisionToUM!killBaseline -> SAFETYLOOP
    []
    fromIO.safe -> (UMtoDecision.complexRunning -> COMPLEXSAFE
                    []
                    BASELINESAFE)
    []
    UMtoDecision.userKillBaseline -> SAFETYLOOP

BASELINESAFE =
    fromBaseline.cntrlout -> toIO!cntrlout -> BASELINELOOP
    []
    ([] x: {illegalout, responseTimeout} @
        fromBaseline.x -> DecisionToUM!killBaseline ->
        fromSafety.cntrlout -> toIO!cntrlout -> SAFETYLOOP)

COMPLEXLOOP =
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    DecisionToUM!killComplex -> TEMPSAFETYLOOP
    []
    fromIO.safe -> COMPLEXSAFE
    []
    UMtoDecision.userKillComplex -> BASELINELOOP

COMPLEXSAFE =
    fromComplex.cntrlout -> toIO!cntrlout -> COMPLEXLOOP
    []
    ([] x: {illegalout, responseTimeout} @
        fromComplex.x -> DecisionToUM!killComplex ->
        BASELINESAFE)

TEMPSAFETYLOOP =
    fromIO.safe -> BASELINESAFE
    []
    fromIO.unsafe -> fromSafety.cntrlout -> toIO!cntrlout ->
    TEMPSAFETYLOOP

--
-- Safety Component:
--
SAFETY =  fromSafety!cntrlout -> SAFETY

--
-- Baseline Component:
--
BASELINE =
    UMtoBaseline.start -> BaselineToUM!initDone ->
    (UMtoBaseline.enableOutput -> BASELINERUNNING
     []
     UMtoBaseline.kill -> BaselineToUM!dead -> BASELINE)

BASELINERUNNING =
    (|~| x: {cntrlout, illegalout, responseTimeout} @
         fromBaseline!x -> BASELINERUNNING)
    []
    UMtoBaseline.kill -> BaselineToUM!dead -> BASELINE

--
-- Complex Component:
--
COMPLEX =
    UMtoComplex.start -> ComplexToUM!initDone ->
    (UMtoComplex.enableOutput -> COMPLEXRUNNING
     []
     UMtoComplex.kill -> ComplexToUM!dead -> COMPLEX)

COMPLEXRUNNING =
    (|~| x: {cntrlout, illegalout, responseTimeout} @
         fromComplex!x -> COMPLEXRUNNING)
    []
    UMtoComplex.kill -> ComplexToUM!dead -> COMPLEX

--
--  Physical I/O component:
--
PHYSICALIO = INPUT ||| OUTPUT

-- Take input from the Plant:
INPUT = fromPlant?status -> fromIO!status -> INPUT

-- Send output to the Plant:
OUTPUT =  toIO?command -> toPlant!command -> OUTPUT


-- +=====================================================+
-- | Properties to verify                                |
-- +=====================================================+

-- Auxiliary definitions:
SIGMA = {| fromUser, fromPlant, toPlant, fromIO, toIO,
           fromComplex, fromBaseline, fromSafety,
           UMtoComplex, UMtoBaseline, ComplexToUM,
           BaselineToUM, DecisionToUM, UMtoDecision,
           convergenceDetected,
           convergenceTimeout,
           raiseBaselinePrio,
           raiseComplexPrio,
           lowerBaselinePrio,
           lowerComplexPrio |}

pragma channel e


-- Property 1. Deadlock-free.
--      |-  P1 is failure-divergence-refined by P1SIMPLEX
-- where:
```

```
P1 = e -> P1

P1SIMPLEX = identify (SIGMA, e, SIMPLEX)

-- To check this property use: Check1 "P1" "P1SIMPLEX";


-- Property 2. Control commands are sent to the plant infinitely often. That
-- is, it is never the case that from a given instant commands are not sent
-- to the plant anymore.
--      |- P2 is failure-divergence-refined by P2SIMPLEX
-- where:

P2 = |~| x: TOPLANT @ toIO!x -> P2

P2SIMPLEX = SIMPLEX \ diff (SIGMA, {| toIO |})

-- To check this property use: Check1 "P2" "P2SIMPLEX";


-- Property 3. It is never the case that there are two consecutive readings
-- from the plant without an output command between them. In other words,
-- after reading the status of the plant, and before performing the next
-- reading, a command has to be sent to the plant. (This could be a way of
-- detecting that a deadline was missed)
--      |-  P3 is trace-refined by P3SIMPLEX
-- where:

P3 = fromIO?x -> toIO!cntrlout -> P3

P3SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, toIO |})

-- To check this property use: CheckTrace "P3" "P3SIMPLEX";


-- Property 4. After unsafe condition the plant is immediately controlled by
-- safety, and as long as it is unsafe it remains in control by safety:
--      |-  P4 is trace-refined by P4SIMPLEX
-- where:

P4 =  fromIO.unsafe -> fromSafety.cntrlout -> toIO.cntrlout -> P4
      |~|
      fromIO.safe -> (|~| c: { fromComplex, fromBaseline, fromSafety } @
                        c.cntrlout -> toIO.cntrlout -> P4)

P4SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, toIO, fromSafety.cntrlout,
                                    fromComplex.cntrlout,
                                    fromBaseline.cntrlout |})

-- To check this property use: CheckTrace "P4" "P4SIMPLEX";


-- Property 5. Whenever the plant is in safe state and is being controlled
-- by the complex controller, if the complex controller does not
-- produce an output on time (i.e. it misses its deadline or falls into an
-- infinite loop) or if the output is illegal, the control of the device is
-- passed to the baseline controller (or to the safety controller, in case
-- the baseline controller fails).
--      |- P5 is trace-refined by P5SIMPLEX
-- where:

P5 = fromIO.safe ->
     (((|~| x: {illegalout, responseTimeout} @
          fromComplex.x -> (|~| c: {fromBaseline, fromSafety} @
                              c.cntrlout -> P5))
       |~|
       (|~| c: {fromComplex, fromBaseline, fromSafety} @ c.cntrlout -> P5)
     )
     |~|
     fromIO.unsafe -> fromSafety.cntrlout -> P5

P5SIMPLEX = SIMPLEX \ diff (SIGMA, {| fromIO, fromComplex,
                                      fromBaseline.cntrlout,
                                      fromSafety.cntrlout |})

-- To check this property use: CheckTrace "P5" "P5SIMPLEX";


-- Property 6. Whenever Complex is started, Baseline has to be running, and
-- it never happens that there is more than one Baseline or more than one
-- Complex running.
--      |- P6 is trace-refined by P6SIMPLEX
-- where:

P6 = UMtoBaseline.start -> P6AUX
P6AUX = UMtoComplex.start -> UMtoComplex.kill -> P6AUX
        |~|
        UMtoBaseline.kill -> P6

P6SIMPLEX = SIMPLEX \ diff (SIGMA, { UMtoBaseline.start, UMtoBaseline.kill,
                                     UMtoComplex.start, UMtoComplex.kill })

-- To check this property use: CheckTrace "P6" "P6SIMPLEX";
```