Component Connection Models of Reaction Control System

Takashi Watanabe and Benjamin Kuipers

September 1993

Component Connection Models of Reaction Control System

Takashi Watanabe *
Nagoya University
and
Benjamin Kuipers †
University of Texas at Austin

January 6, 1994

Abstract

Many of engineering devices consists of components, and component centered modeling presents us a straight way to build global systems of the devices. This modeling is also expected to give representations with minimum redundancy, which are easy to rearrange and are easy to understand for human. We demonstrate component connection models in QSIM and shows its utility.

While we note that the original component connection model (CC) on QSIM has smart ideas, we developed a new model builder which inherits the concepts from CC and has extended facilities in order to perform qualitative and semiquantitative simulations on QSIM. The new model builder (CC-in-algy) is based on Algernon, and almost all the modeling data are held in frames and these data are accessed by inference rules which create QDE definitions. Each component has local constraints in it, and connections of components produces additional constraints between terminals. In order to obtain tractable behaviors, it is, however, sometimes required to use constraints among variables in different components, and it is also necessary to inherit sufficient data from predecessor states at region transitions. CC-in-algy enables us to represent both of them.

As an application of component connections of CC-in-algy, this paper describes the simulation of the space shuttle reaction control system (RCS). Tanks, regulators, pipes and thrusters constitute RCS, and each of them or a group of some of them makes a component which is used to build a global model of RCS.

*School of Informatics and Sciences, Nagoya University, Nagoya 464-01 Japan. E-mail: watanabe@gs.human.nagoya-u.ac.jp

†Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712 USA. Email: kuipers@cs.utexas.edu

Contents

1 Introduction

Component centered modeling [de Kleer and Brown, 1984] utilizes the structure of component elements and predicts the behaviors of global systems. Each component has local representation of its problem and the hierarchical combinations of components are expected to present a total definition of overall problems. The basic idea of component connection is applicable to wide variety of areas, especially to many of industrial devices consisting of parts which are explicitly distinguished from others.

CC-in-algy is a component connection modeller which is developed in order to build QDEs for QSIM [Kuipers, 1986], and is utilized for qualitative and semiquantitative simulations. The original concepts are in CC [Franke and Dvorak, 1990], and CC-in-algy presents some extensions while it follows the ideas of CC. The main part of CC-in-algy is written in Algernon [Kuipers and Crawford, 1993], and almost all the component data are stored in frames and they are processed by forward and backward inference rules in order to retrieve variables, constraints and others which are used to build clauses of QDE definitions of the problems.

In this paper, we apply CC-in-algy to a realistic problem: the simulation of the space shuttle reaction control system (RCS). The qualitative model of RCS is presented by Kay [Kay, 1992] and we remodel RCS in the paradigm of CC-in-algy. There are some problems in the modeling of component connections in QSIM. One of them is a problem in common with almost all component connection models and it is about the necessity of constraints which governs variables in different components. While these constraints are apt to violate the principle of “no function in structure”, simulations on QSIM may sometimes require to introduce them to have a reasonable number of behaviors. Region transitions and the assertion and inheritance of qualitative information and/or semiquantitative information between states at transitions gives another problem. Even if the transitions are found by checking states in local components, it may necessary to refer to states in another components in order to create new states after transitions.

CC-in-algy has a hierarchical component definition in which higher level components include constraints among their variables and variables in lower level components used to build the higher level components. All the transitions in CC-in-algy are realized by the mode feature presented by QSIM. The change of values of mode variables means transitions and the combinations of mode variable values determines which constraints are adopted in the current state. This way of transition is different from that of the original CC in which multiple QDEs are created and a new state uses one of the QDEs. At any level of component hierarchy, the assertion and inheritance information of the component at the level and components lower than the level can be declared in CC-in-algy.

Like many types of engineering systems, RCS includes many components such as tanks, pipes regulators, and it can be seen as a topological combination of components. This paper demonstrates two models of RCS in CC-in-algy. One is the model which is designed to shows almost the same QDE and prediction results as those of Kay’s model. The other model consists of more components while it basically follows the physical interpretation in Kay’s model. The following section describes the overview of the models and the basic components used in the modeling. Section 3 shows the integrated component definitions of RCS. Section 4 gives some examples of simulations and the final section presents comments about future work.

2 Basic Components

2.1 Basic Components of RCS Models in CC-in-algy

Following the abstracted RCS [Kay, 1992], we adopt a system in which the flow from the helium tank to the propellant tank is idealized into a single path. Two models are developed for the simulations of this system.

One model assumes that the RCS consists of four parts of groups of components. The four parts are as follows:

Helium tank

Abstracted regulator with helium valve, secondary regulator and primary regulator

Abstracted fuel tank with propellant tank, tank valve, fuel line and manifold valve

Manifold and Thruster

The other model regards that the RCS has following ten parts:

Helium tank

Helium valve

Secondary regulator

Primary regulator

Fuel tank

Tank valve

Fuel line

Manifold valve

Manifold

Thruster

Hereafter, the model with four parts is called four components model, and the model with ten parts is called ten components model. In both models, each part corresponds to a component called basic component. The definitions of basic components are given in Appendix A.

One special basic component is used in the modeling of RCS. Its name is `space-environment` and the current version of this component holds the value pressure of pressure in the space. This component is used as a subcomponent by other components when the components need the physical values of space environment in order to specify their constraints.

2.2 Basic Components in Four Components Model

In this subsection, we presents a brief comment for each of basic components used in four components model. The four components model is designed to follow the Kay's model. You can, therefore, find the physical interpretation of the model in his report [Kay, 1992].

- One port tank with leak No.1
  Component definition with name `1-Port-Tank-with-Leak#1` is a general one port tank in the space, which may be leaky and is relatively independent from domains of problems. The fluid in the tank is compressible and the flow is inward or outward. This component has a subcomponent `Space-Environment`. The amount of leak is proportional to the pressure difference between the tank and the space environment surrounding the tank. The mode variable `TankLeak?` determines that the leak occurs or not.

- Abstracted regulator
  Component definition with name `Abstracted-Regulator` include the helium valve, the secondary regulator and the primary regulator in RCS. The quantity spaces of variables with type flow says that this component does not allow the reversed flow. That is, the fluid flow is from the valve to the primary regulator. The fluid in this component is assumed incompressible. The mode variable `reg-model` shows the facility of regulator at the current state. The mode variable `reg-mode` shows the mode of the value and/or the fault models of regulators.

- Ullage and liquid tank with tank line
  Component definition with name `Ullage-Liquid-Tank-with-Line` is a tank with a sequential connection of tank line, tank valve and manifold valve. This component assumes the existence of compressible ullage and compressible liquid, and initial, normal and maximum values of their amounts. Gas of ullage is governed by the equation of state. Pressure difference appears across the tank valve, and the amount of flow at outlet is proportional to this pressure difference if all the valves are open. The tank, the tank line and the manifold valve is frictionless. Leak of fluid may occur at the tank and the tank line and it is controlled by the mode variables `TankLeak?` and `LineLeak?`. Mode variables `TankValve` and `ManValve` shows the modes of the tank valve and the manifold valve, respectively.

- Manifold with thruster
  Component definition with name `Manifold-with-Thruster` defines a component with fuel line, manifold and thruster. The manifold is a tank and it accumulates compressible fuel. This component is relatively domain dependent and is designed as a manifold and a thruster of RCS. The pressure at the port knows a regulated pressure value which is denoted by the landmark `PPreg`. Flow leak from a fuel line and/or a manifold is denoted by values of mode variables `LineLeak?` and `ManLeak?`.

2.3 Basic Components in Ten Components Model

In this subsection, we presents a brief comment for each of basic components used in ten components model. Some of basic components in this model are intended to be independent from each other and be independent from the global RCS.

- One port tank with leak No.2
  Component definition with name `1-Port-Tank-with-Leak#2` is a general one port tank with leak, and it is almost same with `1-port-tank-with-leak#1` except that some variables are omitted and the definition is simplified.

- Shut valve without loss
  Component definition with name `Shut-Valve-without-Loss` is a simple shut valve with no loss. When the valve is open and the mode variable `mode` is `open`, the energy loss is zero. When `mode` is `closed`, no flow occurs and pressures at both ports are not constrained and may be indeterministic.

- Shut valve with with loss
  Component definition with name `Shut-Valve-with-Loss` is a shut valve with loss of energy and without any leak. When the valve is open, that is `mode` is `open`, energy loss which is proportional to the flow rate of amount results in the pressure difference across the valve. When the valve is closed, the pressures and, as a consequence, the pressure difference may be indeterministic.

- regulator
  Component definition with name `Regulator` defines a regulator associated with a check valve. The output pressure has a landmark `P-reg` which denotes the regulated pressure value of this regulator. Though the meanings of values of mode variable `reg-model` are same as those of the component definition `abstracted-regulator`, `reg-model` is dynamic [Franke and Dvorak, 1990] and it may change during simulations. When the inflow is not positive, that is, when fluid does not flow in the expected direction, the value of `reg-model` is `closed` and the regulator prevents reverse flow. When the mode variable `reg-mode` is `open`, the regulator is failed open, and when `reg-mode` is `closed`, it is failed closed. The value `regulated` of `reg-mode` shows that the regulator has an ability to regulate its output pressure.

- Gas and liquid tank
  Component definition with name `Gas+Liquid-Tank` is a tank with ullage and liquid. Fluid is compressible and the gas obeys the state equation. The pressure in the tank is uniform. Gas pushes out liquid or vice versa. Maximum amounts, minimum amounts and normal amounts of gas and liquid are prescribed. Leak of liquid may occur when the mode variable `Leak?` is `yes`.

- Pipe with leak and no loss
  Component definition with name `Pipe-with-Leak-and-no-Loss` defines a leaky pipe through which compressible fluid flows. While the pressure is uniform and there is no pressure loss, leak may occur.

- two port tank with leak
  Component definition with name `2-Port-Tank-with-Leak` gives a two port version of a one port tank `1-port-tank-with-leak#2`. No flow directions are assumed and the tank accumulates or emits fluids.

- Thruster
  Component definition with name `Thruster` is a general thruster of rocket engines. It does not have any prescribed value of pressure, but has only the physical relations between variables.

3 Integrated Components: RCS Models

3.1 RCS Models

We now use basic components and build two integrated components in order to predict the behaviors of RCS. In both models, basic components are sequentially connected and each connection nodes have two terminals.

As we will see in the following subsections, the integrated components have some descriptions which can be regarded as violations of the principle of "no function in structure" [de Kleer and Brown, 1984]. One of the explicit forms of them is that of constraints. In order to obtain physically meaningful predictions, or in order to have a tractable number of behaviors, we may impose additional constraints which govern variables in different subcomponents. Another violation of the principle appears as landmarks in quantity spaces. For example, the pressure of a general one port tank is not expected to have values required to drive regulators. When we use this tank as a helium tank of RCS, we may need to introduce designed values as landmarks in quantity space of pressure. That is, additional landmark may be used when the integrated model is built. The information to be inherited at region transitions also depends on the connections among components and the overall problem.

In the demonstrations shown here, all of these descriptions are declared in clauses of the higher level components. Definitions of the integrated components are given in Appendix B.

3.2 RCS Four Components Model

Component definition with name `RCS-component#1` defines a component of four components model. As you can see in the `connections` clause of the definition, this component has four subcomponents and they are connected in series.

In `components` clause, while the component definitions `Abstracted-Regulator` and `Manifold-with-Thruster` are instantiated, without any change of definitions, as components `Regulator` and `Manifold+Thruster`, respectively, the components `He-Tank` which is an instantiation of `1-Port-Tank-with-Leak-#1` and `Fuel-Tank` which is an instantiation of `Ullage-Liquid-Tank-with-Line` are associated with the declarations of quantity spaces for some variables. The quantity space for flow `Flow` of `He-Tank`, for example, restricts that the flow direction of the helium tank is only outward, and the quantity space for the pressure `P` distinguishes the minimum value `PHeMin`, value to drive a primary regulator `PHePreg`, value to drive a secondary regulator `PHeSreg` and the initial value `PHeInit`. The landmarks in these quantity spaces are introduced as a consequence of component connections.

The component definition `RCS-components#1` has a dynamic mode variable `operation`, and all the region transition of this model is controlled by the mode value conditions. Mode value specifications include the information about assertions and inheritances of values, and allow QSIM to carry out reasonable reasoning. The value of `operation` is `initial-state` at the beginning of simulations.

Some of the constraint specifications in `constraints` clauses are those which concerns with variables in different components. Even though the constraints have variables in same components, their constraint specifications are declared in the higher level components, if they are moded constraints and the mode conditions of them include mode variables in different components. Some other constraints have the same forms with those in basic components, but they are associated with different corresponding values. These corresponding values appear because the quantity spaces given in the higher level components include additional landmarks which specify problems and need to be related as corresponding values. For example, the constraint specification in `1-Port-Tank-with-Leak#1`

```
((M+ Amt P))
```

is redefined, by using new landmarks, as

```
((m+ (He-Tank Amt) (He-Tank P)) (0 0) (AmtHeMin PHeMin) (AmtHePreg PHePreg)
                                (AmtHeSreg PHeSreg) (AmtHeInit PHeInit))
```

3.3 RCS Ten Components Model

Component definition with name `RCS-components#2` defines a component of ten components model as a serial connection of ten basic components. In this model, the fuel tank does not directly connect to the fuel tank, but it exists between the tank valve and the manifold valve.

The landmark `PPreg` of `Fuel-Tank` corresponds to the regulated pressure of the primary regulator, and `SPReg` corresponds to the regulated pressure of the secondary regulator when no pressure difference appears across the primary regulator.

Some components have same component definitions, but they are instantiated independently. Quantity spaces of pressure are redeclared, because the pressures in basic components are strongly related.

Subcomponents `S-Regulator` and `P-Regulator` are instantiations of `Regulator`, and each of them has a dynamic mode variable `reg-model`. Though region transitions occur when mode value conditions of these variables are satisfied, these subcomponents may not have enough inheritance and assertion data about transitions of a global system. The mode variable `operation` in `RCS-components#2` has mode values with mode value conditions which are satisfied if conditions in regulator components are satisfied, and it provides enough information about transitions of the total model.

The declarations of equivalences between landmarks introduced in the quantity spaces of the `components` clause may be required. Two ways are used to declare equivalences in this model. One is the way which specifies the partial order among landmarks and introduces = constraints between variables. In `qspace-hierarchy` clause, the hierarchical quantity space of pressures in the helium tank, helium valve, regulators and the fuel tank is held. This hierarchical quantity space is referred to via = constraints in the `constraints` clause. Other way is to use `equal` constraints and declare the equivalence of landmarks as corresponding values in constraint specifications. The last constraint specification, for example, gives the equivalence between landmarks of the pressure in the manifold and the pressure in the thruster.

4 Simulations of RCS Models

4.1 Simulation Environment

4.1.1 Simulation Function

Both of the models have not a few variables, and some of values of these variables need to be asserted initially for reasonable simulations. Kay [Kay, 1992] developed the function `run-rcs` which accepts the description of physical state of RCS, translates it into the representations for the RCS model, and processes simulation. We introduce the functions `run-rcs#1` and `run-rcs#2` for the four components model and the ten components model, respectively. These function have same syntax as that of `run-rcs`, and they are used in order to obtain sample results.

4.1.2 QSIM Tuning Conditions

QSIM provides many facilities in order to create reasonable behaviors for its simulations. In the simulations presented here, following declarations are done in order to reduce the number of branchings of behaviors.
Four components model

```
:ignore-qdirs
  (He-Tank dAmt) (He-Tank Flow) (He-Tank PDiffVac) (He-Tank TotalLeak)
  (He-Tank OutFlow) (He-Tank MFlow) (Regulator Flow-i) (Regulator Flow-o)
  (Fuel-Tank dAmtLiquid) (Fuel-Tank PDiff) (Fuel-Tank Flow-i)
  (Fuel-Tank Flow-o) (Fuel-Tank OutletFlow) (Fuel-Tank LiquidOutFlow)
  (Fuel-Tank TotalLeak) (Fuel-Tank LineLeak) (Fuel-Tank TankLeak)
  (Manifold+Thruster Den*PDiff) (Manifold+Thruster f[Den*PDiff])
  (Manifold+Thruster Flow) (Manifold+Thruster Man->ThFlow)
  (Manifold+Thruster PDiff) (Manifold+Thruster dAmt)
  (Manifold+Thruster MOutflow) (Manifold+Thruster Outflow)
  (Manifold+Thruster TotalLeak) (Manifold+Thruster LineLeak)
  (Manifold+Thruster ManLeak)
:no-new-landmarks
  (He-Tank dAmt) (He-Tank Flow) (He-Tank Outflow) (He-Tank Mflow)
  (Regulator Flow-i) (Regulator Flow-o) (Fuel-Tank PDiffVac)
  (Fuel-Tank PDiff) (Fuel-Tank Flow-i) (Fuel-Tank Flow-o)
  (Fuel-Tank OutletFLow) (Fuel-Tank dAmtLiquid) (Fuel-Tank LiquidOutflow)
  (Fuel-Tank TotalLeak) (Manifold+Thruster PDiffVac)
  (Manifold+Thruster Flow) (Manifold+Thruster dAmt)
  (Manifold+Thruster Man->ThFlow) (Manifold+Thruster Outflow)
  (Manifold+Thruster MOutflow) (Manifold+Thruster TotalLeak)
  (Manifold+Thruster LineLeak)
```

Ten components model

```
:ignore-qdirs
  (He-Tank dAmt) (He-Tank Flow) (He-Tank PDiffEnv) (He-Tank LeakFlow)
  (He-Valve Flow1) (He-Valve Flow2) (S-Regulator Flow-i)
  (S-Regulator Flow-o) (P-Regulator Flow-i) (P-Regulator Flow-o)
  (Fuel-Tank dAmtLiquid) (Fuel-Tank FLowG) (Fuel-Tank FLowL)
  (Fuel-Tank LeakFlow) (Tank-Valve FLow1) (Tank-Valve FLow2)
  (Tank-Valve PDiff) (Fuel-Line Flow1) (Fuel-Line Flow2)
  (Fuel-Line SumFlow) (Fuel-Line dAmt) (Fuel-Line LeakFlow)
  (Man-Valve Flow1) (Man-Valve Flow2)  (Manifold Flow1) (Manifold Flow2)
  (Manifold SumFlow) (Manifold dAmt) (Manifold LeakFlow) (Thruster Flow)
  (Thruster PDiff) (Thruster Den*PDiff) (Thruster f[Den*PDiff])
:no-new-landmarks
  (He-Tank dAmt) (He-Tank Flow) (He-Valve Flow1) (He-Valve Flow2)
  (S-Regulator Flow-i) (S-Regulator Flow-o)  (P-Regulator Flow-i)
  (P-Regulator Flow-o) (Fuel-Tank FlowG) (Fuel-Tank FlowL)
  (Fuel-Tank PDiffEnv) (Fuel-Tank dAmtLiquid) (Tank-Valve Flow1)
  (Tank-Valve FLow2) (Tank-Valve PDiff) (Fuel-Line FLow1)
  (Fuel-Line Flow2) (Fuel-Line SumFlow) (Fuel-Line dAmt)
  (Fuel-Line LeakFlow) (Fuel-Line PDiffEnv)
  (Man-Valve Flow1) (Man-Valve Flow2) (Manifold Flow1)  (Manifold Flow2)
  (Manifold SumFlow) (Manifold dAmt) (Manifold LeakFlow)
  (Manifold PDiffEnv) (Thruster Flow)
:unreachable-values
  ((Fuel-Tank P) PMax) ((Fuel-Line P) PMax) ((Thruster P)  PMax)
```

4.1.3 Initial States

In the steady normal state of the four components model, the thruster is off and, if the helium valve is open, `reg-model` of `Regulator` component is `ideal-psource` and `reg-mode` is `preg` when the pressure in `He-Tank` is higher than `PHePreg`. The pressure in the fuel tank is regulated to be `PUllPreg` and the amounts and volumes of ullage and fuel are normal values denoted by the suffix `Nom`. No flow occurs in the steady normal state and the pressures in the fuel pipe and the manifold are also regulated.

In the steady normal state of the ten components model, the thruster is off and both of `reg-modes` of `S-Regulator` and `P-Regulator` are `regulated`. If the helium valve is open, `reg-models` are `ideal-psource` when the pressure in `He-Tank` is above `PHeSreg`, and `reg-model` of `S-Regulator` is `invisible` and `reg-model` of `P-Regulator` is `ideal-psource` when the pressure in `He-Tank` is between `PHePreg` and `PHeSreg`. The pressure in the fuel tank is `PPreg` and the amounts and volumes of ullage and fuel are normal values.

Two kinds of initial states are adopted in the following example simulations. One is the initial states of the simulations which start from a perturbed steady normal state. The perturbations are physical effects of valve openings, sudden beginnings of leaks and breakdowns of regulators. This kind of initial states are created by inheriting pressures and the amount of fuel in the fuel tank from a steady normal state. The other initial state is a state where the pressures in the fuel tank and the manifold are lower than the values regulated by the primary regulator pressure and `reg-model` of the primary regulator is, therefore, `pipe`, and the amount of ullage in the fuel tank is between its initial value and its normal value, while the volumes of ullage and fuel in the fuel tank are normal and the pressure difference across regulators is positive.

4.2 Examples

4.2.1 Four Components Model

The four components model is intended to present the same behaviors as those of Kay's model [Kay, 1992], and it works as it is expected. (Note: There may be some differences in behaviors as of September 27, 1993. These differences simply occurs because of a tiny bug in QSIM.)

As an example of the performance given by the four components model, Figure 1 shows one of the behaviors of RCS in which all the components are normally working and the thruster is on.

4.2.2 Ten Components Model

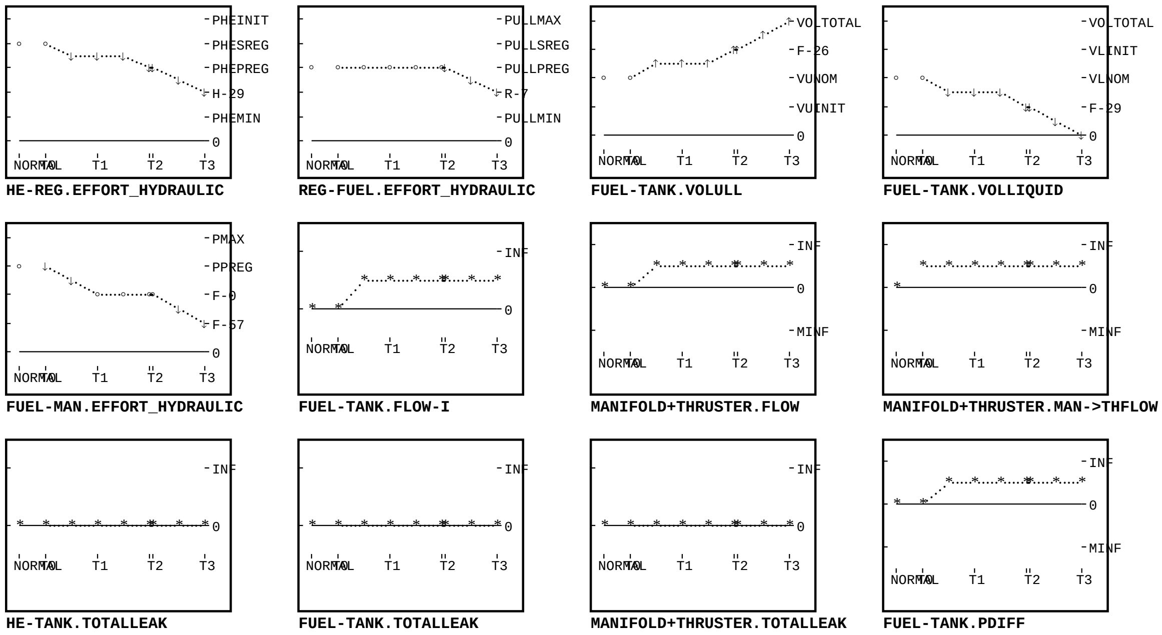
Because the physical interpretations of the ten components model are not exactly same with the four components model's, the predictions may be also different from those of the four components model.

Follows are the examples which corresponds with those in [Kay, 1992].

- When all the components are normally working and the thruster is on, the ten components model gives the same behaviors with those of the four components.
- In the case that the helium tank begins to leak, the behavior of RCS which is initially at the steady normal state is predicted as shown in Figure 2. Because the primary regulator closes, no reverse flow from the fuel tank to the helium tank occurs, even though the pressure difference across the primary regulator is negative.
- For a leaky fuel line, the simulation which starts from the steady normal state with a closed manifold valve shows a behavior in Figure 3. In this simulation, the fuel tank is only the fluid source of the fuel line and the pressure difference across the tank valve remains positive, while the pressures in the fuel tank and the fuel pipe are monotonically decreasing.
- Figure 4 shows the result of the case that the primary regulator becomes failed open. The initial state is a steady normal state with the helium tank pressure `PHeInit`. The pressure of the fuel tank increases and approaches to `PSreg`. Unless the flow through regulators is zero, the primary regulator with `pipe` model requires a pressure difference and the pressure in the helium tank does not reach `PSreg`.
- The behavior starting from low pressure in the fuel tank is shown in Figure 5. Low pressure in the fuel tank initially makes the primary regulator `pipe`.

The above simulations except one in Figure 2 have multiple behaviors. Many of the branchings occur because landmarks of pressures and amounts are prescribed and their correspondences are not well defined.

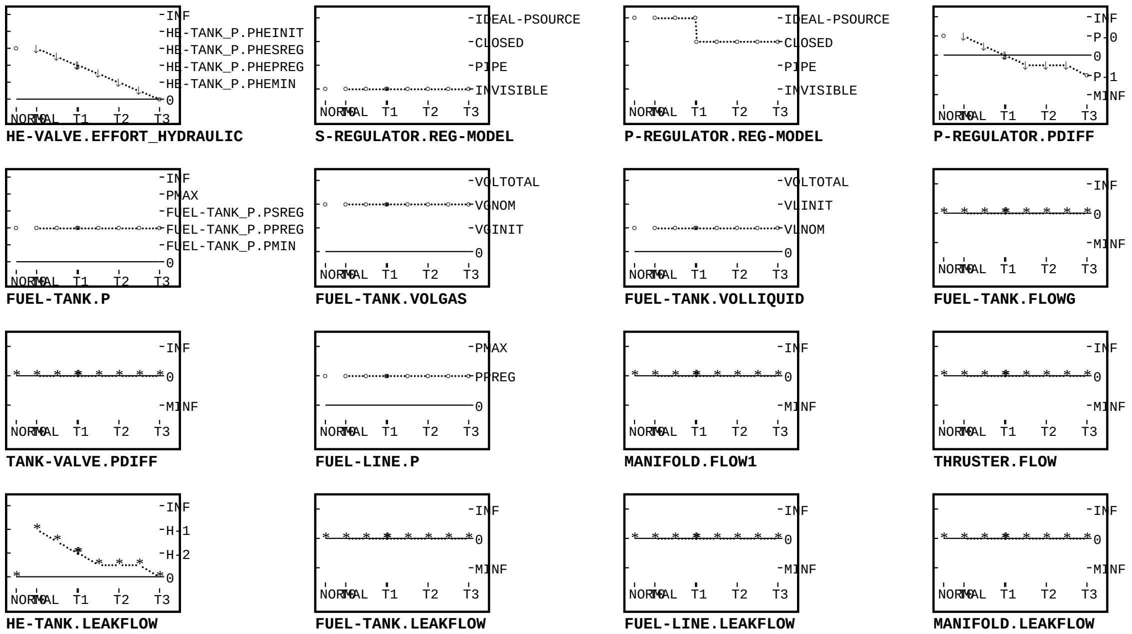
Structure: RCS-COMPONENT#1_QDE.
Behavior 6 of 15: (S-1 S-2 S-6 S-12 S-16 S-23 S-29 S-32).
Final state: (TRANSITION FINAL GF COMPLETE), (TRANSITION-IDENTITY), T<INF.




```
(run-rcs#1 :init 'equilibrium
           :phe  'phesreg
           :configuration  '((leg1 closed working working)
                             (leg2 open working working)
                             (tankvalve open) (manvalve open)
                             (thruster max)))
```

Figure 1: Four components model simulation of RCS with a thruster on and no fault in components. (Corresponds with Figure 6 in [Kay 1992].)

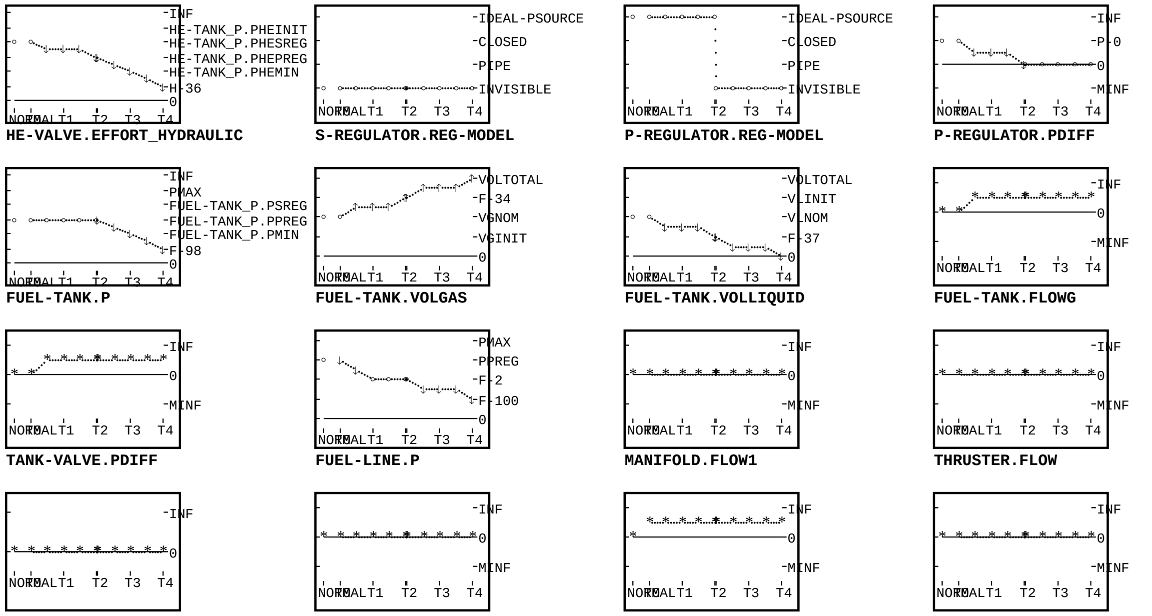
Structure: RCS-COMPONENT#2_QDE.
Behavior 1 of 1: (S-1 S-2 S-3 S-4 S-5 S-6 S-7 S-8).
Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.




```
(run-rcs#2 :init 'equilibrium
           :phe 'phesreg
           :configuration '((leg1 closed working working)
                            (leg2 open working working)
                            (hetankleakhyp yes) (heleaks max)
                            (tankvalve open) (manvalve open)
                            (thruster 0)))
```

Figure 2: Ten components model simulation of a helium tank leak with a thruster off. (Corresponds with Figure 8 in [Kay 1992].)

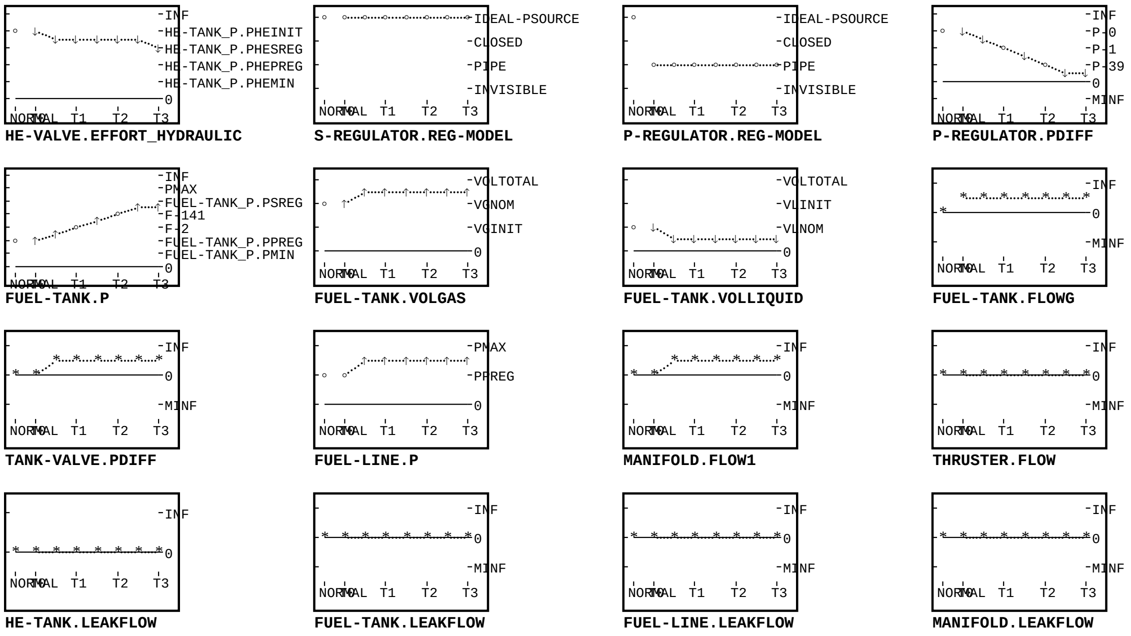
Structure: RCS-COMPONENT#2_QDE.
Behavior 7 of 15: (S-1 S-2 S-6 S-12 S-16 S-23 S-29 S-33 S-36 S-37).
Final state: (TRANSITION FINAL GF COMPLETE), (TRANSITION-IDENTITY), T<INF.




```
(run-rcs#2 :init 'equilibrium
           :phe 'phesreg
           :configuration '((leg1 closed working working)
                            (leg2 open working working)
                            (tanklineleakhyp yes) (lineleaks max)
                            (tankvalve open) (manvalve closed)
                            (thruster 0)))
```

Figure 3: Ten components model simulation of a fuel line leak with a closed manifold valve.

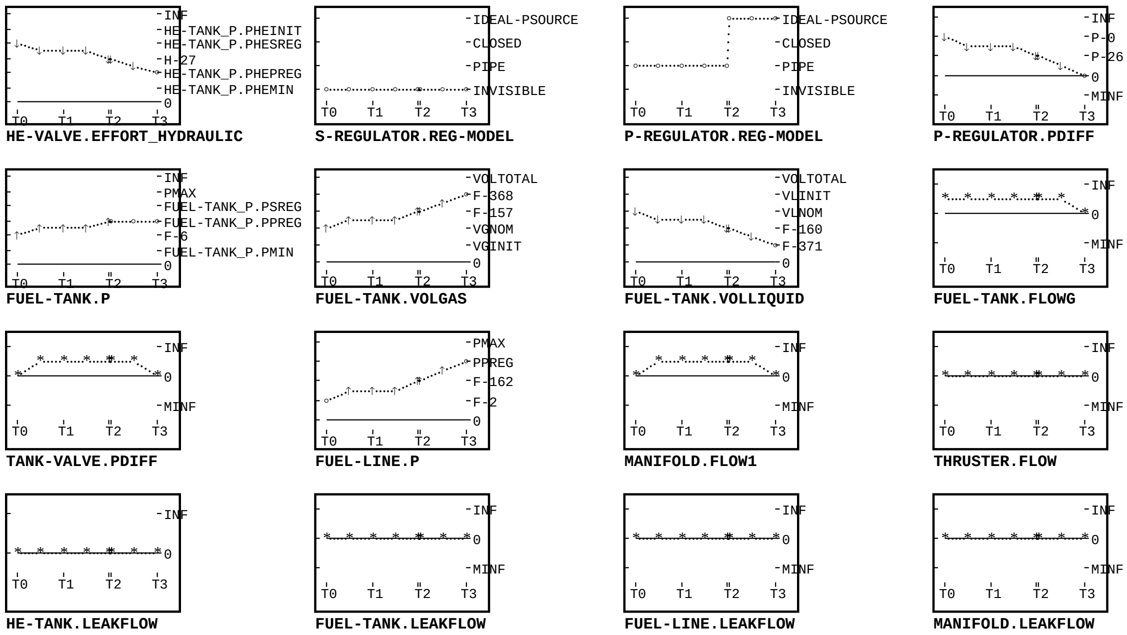
Structure: RCS-COMPONENT#2_QDE.
Behavior 62 of 87: (S-1 S-2 S-6 S-30 S-59 S-100 S-206).
Final state: (GF COMPLETE), (NIL), T<INF.




```
(run-rcs#2 :init 'equilibrium
           :phe 'pheinit
           :configuration '((leg1 closed working working)
                            (leg2 open failed-open working)
                            (tankvalve open) (manvalve open)
                            (thruster 0)))
```

Figure 4: Ten components model simulation with a failed open primary regulator

Structure: RCS-COMPONENT#2_QDE.
Behavior 15 of 65: (S-0 S-1 S-3 S-9 S-15 S-42 S-55 S-81).
Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.



(run-rcs#2 init 'low-pull)

Figure 5: Ten components model simulation starting from a fuel tank with low pressure

5 Conclusion

Many of engineering problems consist of components. CC-in-algy is a component connection model builder for QSIM. In this paper, we have applied CC-in-algy to the space shuttle reaction control system and showed that reasonable result is obtained.

Some of future works suggested are as follows:

> The current basic components are somewhat ad hoc and may not be powerful enough to predict behaviors of more realistic problems. Not only the understanding of physical phenomena of components but also the practical application seem to make component definitions more sophisticated.
>
> In addition to landmarks in each components, some additional landmarks are introduced at higher level components in order to specify global problems. The introduction of new landmarks requires the declarations of corresponding values among them, and it, therefore, needs to add some constraint specifications to the higher level components, even though these constraints of constraint specifications with less number of corresponding values are already defined in lower level components. In the current models, most of the corresponding values are specified by the programmer. QSIM has an ability to create landmarks by itself, and we can expect QSIM to introduce necessary landmarks so long as we define the least number of landmarks. In this case, we also need to develop powerful and restrictive constraints which suppress branching.
>
> The current CC-in-algy applies the Kirchhoff's laws to the variables whose generic type is effort and flow in the bond graph notation. That is, in the hydraulic domain used in the demonstrations, the laws are applied to the flow of fluid and the pressure. In more detail problems, human intuition may assumes the common values not only of pressures but also velocity heads and position heads at the junction of hydraulic components. The more flexible usage of the Kirchhoff's laws are required in these problems.

The code of CC-in-algy, basic component library and the RCS models should be available from the authors.

References

[de Kleer and Brown, 1984] de Kleer J., Brown J.S., "A Qualitative Physics Based on Confluences", Artificial Intelligence, Vol.24 (1984), 7-83.

[Farquhar et al, 1991] Farquhar, A., Kuipers, B., Rickel, J., Throop, D. and The Qualitative Reasoning Group, 'QSIM: The Program and its Use", Technical Report AI90-123, Artificial Intelligence Laboratory, University of Texas at Austin, Texas, 78712, October 1991.

[Franke and Dvorak, 1990] Franke D., Dvorak D., "CC: Component Connection Model for Qualitative Simulation – A User's Guide", Technical Report AI90-126, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas, 78712, January 1990.

[Karnopp et al. 1990] Karnopp, D., Margolis and D., Rosenberg, R., System Dynamics – A Unified Approach 2nd ed., John Wiley & Sons Inc., New York, 1990.

[Kay, 1992] Kay H., "A Qualitative Model of the Space Shuttle Reaction Control System", Technical Report AI92-188, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas, 78712, September 1992.

[Kuipers, 1986] Kuipers J.B., "Qualitative Simulation", Artificial Intelligence, Vol.29 (1986), 289-338.

[Kuipers, 1992] Kuipers, B., "Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge", APS CS395T, Texas Union Copy Center, University of Texas at Austin, Austin, Texas, 78712, August, 1992.

[Kuipers and Crawford, 1993] Kuipers, B. and Crawford J., "Short Algernon Reference Manual for Algernon Version 1.3.0", unpublished manuscript, January, 1993.

A Basic Component Definitions

```
;;; -*- Syntax: Common-Lisp; Package: USER -*-

(in-package :user)

;;;  Basic CC component library for the Reaction Control System in
 CC-in-algy.

;;#####################################################################
;;;                   Universal Basic Components                    ###
;;;####################################################################

;;;-------------------------------------------------------------------
;;;  CC-component:  Space-environment
;;;  defines space environment in hydraulic domain.
;;;  Each component which uses data in space environment has
;;;  Space-environment component as its subcomponent.  It is somewhat
;;;  curious, because the space-environments of CC components are not
;;;  different, but are same.  That is, an environment is unique for
;;;  constituting components and  may be a kind of super components
;;;  of them.  CC-component, however, does not have any way to access
;;;  data in its super component.  Is there any smart way to represent
;;;  "environment of the world of model"?
;;;-------------------------------------------------------------------

(define-component Space-Environment hydraulic
  "Space: the final frontier"
  (component-variables (P pressure (quantity-space (0 inf))))
  (constraints ((constant P 0))))



;;;####################################################################
;;;           Basic Components for Four Components Model            ###
;;;####################################################################

;;;-------------------------------------------------------------------
;;;  CC-component:  1-Port-Tank-with-Leak#1
;;;                 (tank with one outlet or inlet)
;;;          +--------------+
;;;          |              |
;;;          |              |=
;;;          |              |
;;;          +--------------+
;;;  Fluid leaks from tank shell.
;;;  This component is a basic component.  No quantity-spaces is given
;;;  to Flow, TotalLeak, LeakCoef, Outflow or Mflow.  This implies
;;;      flow may be from tank or into tank
;;;      leak may be from tank or into tank
;;;-------------------------------------------------------------------

(define-component 1-Port-Tank-with-Leak#1 hydraulic
  "one port tank"
```

```
  (terminal-variables (port  (Flow flow)
                             (P pressure)))
  (component-variables (Amt       amount (quantity-space (0 inf)))
                       (dAmt      void)
                       (TotalLeak flow)
                       (LeakCoef  void (quantity-space (0 max)))
                       (PDiffVac  pressure (quantity-space (0 inf)))
                       (Outflow   flow)
                       (MFlow     flow))  ;; minus flow
  (mode-variables (TankLeak? yes no))
  (components (Space Space-Environment))
  (constraints ((d/dt Amt dAmt))
               ((minus dAmt Outflow))
               ((minus Flow MFlow))
               ((M+ Amt P))
               ((add MFlow TotalLeak Outflow))
               ((mult LeakCoef PDiffVac TotalLeak))
               ((add PDiffVac (Space P) P))
               ((TankLeak? yes) -> ((constant Leakcoef max)))
               ((TankLeak? no)  -> ((zero-std Leakcoef)))))

;;;------------------------------------------------------------------
;;;  CC-component:  Abstracted-Regulator
;;;  High and low pressure regulators with shut valve.
;;;                     +----+     +----+
;;;            ___    __|__  |   __|__  |
;;;           __|__   |_|_|  |   |_|_|  |
;;;          =|_|_|===|___|======|___|====
;;;           Valve   High Reg   Low Reg
;;;                  (secondary) (primary)
;;;  No leak is expected.
;;;------------------------------------------------------------------

(define-component Abstracted-Regulator hydraulic
  "regulator"
  (terminal-variables (inlet  (Flow-i flow (quantity-space (0 inf)))
                              (Pin    pressure (quantity-space (0 inf))))
                      (outlet (Flow-o flow (quantity-space (minf 0)))
                              (Pout   pressure (quantity-space (0 inf)))))
  (component-variables (PDiff  pressure))
  (mode-variables (reg-model ideal-psource invisible closed pipe)
                  (reg-mode  open sreg preg closed))
  (constraints ((add PDiff Pout Pin))
               ((minus Flow-i Flow-o))
               ((reg-model pipe)
                -> ((m+ PDiff Flow-i) (0 0) (inf inf)))
               ((reg-model closed)
                -> ((zero-std Flow-i)))
               ((and (reg-model ideal-psource)
                     (or (reg-mode preg) (reg-mode sreg)))
                -> ((constant Pout)))
               ((reg-model invisible)
                -> ((m+ Pin Pout) (0 0))
                   ((zero-std PDiff)))))
```

```
;;;-------------------------------------------------------------------
;;;  CC-component:  Ullage-Liquid-Tank-with-Line
;;;          +-----------------+           ---     ---
;;;          |        #        |         __|__   __|__  outlet
;;;    inlet =| Ullage # Liquid |===========|_|_|===|_|_|==
;;;          |        #        |   line    Tank    Manifold
;;;          +-----------------+           Valve   Valve
;;;  Fluid leaks from tank shell and line wall.
;;;  Flows from inlet to outlet.
;;;-------------------------------------------------------------------

(define-component Ullage-Liquid-Tank-with-Line hydraulic
  "Liquid Tank with Ullage"
  (terminal-variables (inlet-side (Flow-i flow
                                        (quantity-space (0 inf)))
                                  (InletP pressure
                                        (quantity-space (0 PUllMax))))
                      (outlet-side  (Flow-o  flow)
                                    (OutletP pressure)))
  (component-variables (AmtUll amount
                               (quantity-space
                                 (0 AmtUllInit AmtUllNom AmtUllMax)))
                      (VolUll amount
                               (quantity-space (0 VUInit VUNom VolTotal)))
                      (nRT[AmtUll] void
                                    (quantity-space
                                      (0 nRTInit nRTNom nRTMax)))
                      (AmtLiquid amount
                                  (quantity-space
                                    (0 AmtLNom AmtLInit AmtLMax)))
                      (VolLiquid amount
                                  (quantity-space
                                    (0 VLNom VLInit VolTotal)))
                      (VolTotal amount (quantity-space (VolTotal)))
                      (DenLiquid void (quantity-space (0 DLMax)))
                      (dAmtLiquid void)
                      (PDiff pressure)
                      (ValveFlowCoef void (quantity-space (0 max)))
                      (TotalLeak flow (quantity-space (0 inf)))
                      (TankLeak flow (quantity-space (0 inf)))
                      (LineLeak flow (quantity-space (0 inf)))
                      (TankLeakCoef void (quantity-space (0 max)))
                      (LineLeakCoef void (quantity-space (0 max)))
                      (PDiffVac pressure (quantity-space (0 inf)))
                      (LiquidOutflow flow (quantity-space (0 inf)))
                      (OutletFlow flow))
  (mode-variables (TankLeak?   yes no)
                  (LineLeak?   yes no)
                  (TankValve   open closed)
                  (ManValve    open closed))
  (components (Space Space-Environment))
  (constraints ((d/dt AmtUll Flow-i))
               ((d/dt AmtLiquid dAmtLiquid))
               ((minus OutletFlow Flow-o))
               ((minus LiquidOutFlow dAmtLiquid))
```

```
            ((add PDiff OutletP InletP))
            ((mult PDiff ValveFlowCoef OutletFlow))
            ((and (TankValve open) (ManValve open))
             -> ((constant ValveFlowCoef max)))
            ((or (TankValve closed) (ManValve closed))
             -> ((zero-std ValveFlowCoef)))
            ((add TotalLeak OutletFlow LiquidOutFlow))
            ((add TankLeak LineLeak TotalLeak))
            ((add PDiffVac (Space P) InletP))
            ((mult TankLeakCoef PDiffVac TankLeak))
            ((and (LineLeak? yes) (TankValve open))
             -> ((mult LineLeakCoef PDiffVac LineLeak)))
            ((or (LineLeak? no) (TankValve closed))
             -> ((zero-std LineLeak)))
            ((TankLeak? yes)
             -> ((constant TankLeakCoef max)))
            ((TankLeak? no)
             -> ((zero-std TankLeakCoef)))
            ((m+ AmtUll nRT[AmtUll]) (0 0) (AmtUllinit nRTInit)
                                    (AmtUllNom nRTNom)
                                    (AmtUllMax nRTMax))
            ((mult InletP VolUll nRT[AmtUll]) (0 0 0)
                                             (PUllMax VolTotal nRTMax))
            ((m+ InletP DenLiquid) (0 0) (PULLmax DLMax))
            ((mult DenLiquid VolLiquid AmtLiquid))
            ((add VolLiquid VolUll VolTotal) (VolTotal 0 VolTotal)
                                            (0 VolTotal VolTotal)
                                            (VLNom VUNom VolTotal)
                                            (VLInit VUInit VolTotal))
            ((LineLeak? yes)
             -> ((constant LineLeakCoef max)))
            ((LineLeak? no)
             -> ((zero-std LineLeakCoef)))
            ((constant VolTotal))))

;;;------------------------------------------------------------------
;;;  CC-component:  Manifold-with-Thruster
;;;                             /
;;;               +----------+ /
;;;         ======| Manifold |
;;;       Line    +----------+ \  Thruster
;;;                             \
;;;  Fluid leaks from line wall and manifold shell.
;;;  Manifold is designed to have a specified (or regulated) amount of
;;;  liquid  "Amtpreg" at a specified (or regulated) pressure "Preg".
;;;
;;;  In the original RCS model in
;;;        /u/bert/ames/rcs/tech-report/rcs-model.lisp,
;;;  mode variable "TankLineLeakHyp" controls the leak not only a line
;;;  attached to the fuel tank, but also controls the leak a line
;;;  connected to manifold.  In this CC model, a new mode variable
;;;  "LineLeak" is used to control the leak from line to manifold.
;;;  Do not forget to assert values of LineLeaks in both
;;;  "Ullage-Liquid-Tank-with-Line" component and
;;;  "Manifold-with-Thruster" component.
```

```
;;;-------------------------------------------------------------------

(define-component Manifold-with-Thruster hydraulic
  "Manifold and Thruster"
  (terminal-variables (port (Flow flow)
                            (P    pressure
                                  (quantity-space (0 PPreg PMax)))))
  (component-variables (Amt amount (quantity-space (0 Amtpreg AmtMax)))
                       (dAmt void)
                       (Man->ThFlow flow)
                       (PDiff pressure (quantity-space (minf 0 PD* inf)))
                       (DenLiquid void (quantity-space (0 inf)))
                       (Den*PDiff void (quantity-space (0 inf)))
                       (f[Den*PDiff] void (quantity-space (0 inf)))
                       (Thruster void (quantity-space (0 Max)))
                       (TotalLeak flow (quantity-space (0 inf)))
                       (ManLeak flow (quantity-space (0 inf)))
                       (LineLeak flow (quantity-space (0 inf)))
                       (ManLeakCoef void (quantity-space (0 Max)))
                       (LineLeakCoef void (quantity-space (0 Max)))
                       (PDiffVac pressure (quantity-space (0 inf)))
                       (Outflow flow)
                       (MOutflow flow)) ;; minus Outflow
  (mode-variables (LineLeak? yes no)
                  (ManLeak? yes no))
  (components (Space Space-Environment))
  (constraints ((d/dt Amt dAmt))
               ((m+ Amt P) (0 0) (Amtpreg PPreg) (AmtMax PMax))
               ((add PDiff (Space P) P) (PD* 0 PPreg))
               ((mult PDiff DenLiquid Den*PDiff))
               ((m+ P DenLiquid) (0 0) (PMax inf))
               ((m+ Den*PDiff f[Den*PDiff]) (0 0) (inf inf))
               ((mult Thruster f[Den*PDiff] Man->ThFlow))
               ((add Flow MOutflow dAmt))
               ((w+ PDiff Den*PDiff))
               ((add TotalLeak Man->Thflow Outflow))
               ((minus MOutflow Outflow))
               ((add ManLeak LineLeak TotalLeak))
               ((add PDiffVac (Space P) P))
               ((mult ManLeakCoef PDiffVac ManLeak))
               ((LineLeak? yes)
                -> ((mult LineLeakCoef PDiffVac LineLeak)))
               ((LineLeak? no)
                -> ((zero-std LineLeak)))
               ((ManLeak? yes)
                -> ((constant ManLeakCoef max)))
               ((ManLeak? no)
                -> ((zero-std ManLeakCoef)))
               ((LineLeak? yes)
                -> ((constant LineLeakCoef max)))
               ((LineLeak? no)
                -> ((zero-std LineLeakCoef)))
               ((constant Thruster))))
```

```
;;;#####################################################################
;;;            Basic Components for Ten Components Model           ###
;;;#####################################################################

;;;-------------------------------------------------------------------
;;;  CC-component:  1-Port-Tank-with-Leak#2
;;;                 (tank with one outlet or inlet)
;;;  In 1-Port-Tank-with-Leak#1, some auxiliary variables
;;;  (Outflow, Mflow) are used.  1-Port-Tank-with-Leak#2 has somewhat
;;;  sophisticated formulation without these variables.
;;;-------------------------------------------------------------------

(define-component 1-Port-Tank-with-Leak#2 hydraulic
  "one port tank"
  (terminal-variables (port  (Flow flow)
                             (P pressure)))
  (component-variables (Amt       amount)
                       (dAmt      void)
                       (Leakflow  flow)
                       (LeakCoef  void (quantity-space (0 LC)))
                       (PDiffEnv  pressure))
  (mode-variables (Leak? yes no))
  (components (Environment Space-Environment))
  (constraints ((d/dt Amt dAmt))
               ((M+ Amt P))
               ((mult LeakCoef PDiffEnv LeakFlow))
               ((add LeakFlow dAmt Flow))
               ((add PDiffEnv (Environment P) P))
               ((Leak? yes) -> ((constant Leakcoef LC)))
               ((Leak? no)  -> ((zero-std Leakcoef)))))

;;;-------------------------------------------------------------------
;;;  CC-component:  Shut-Valve-without-Loss
;;;         ___
;;;       __|__
;;;      =|_|_|==
;;;  Fluid flows to both directions.
;;;-------------------------------------------------------------------

(define-component Shut-Valve-without-Loss hydraulic
  "Shut Valve without Loss"
  (terminal-variables (port1 (Flow1 flow)
                             (P1    pressure))
                      (port2 (Flow2 flow)
                             (P2    pressure)))
  (mode-variables (mode open closed))
  (constraints ((mode open)
                -> ((minus Flow1 Flow2))
                   ((equal P1 P2)))
               ((mode closed)
                -> ((zero-std Flow1))
                   ((zero-std Flow2)))))

;;;-------------------------------------------------------------------
```

```
;;;  CC-component:  Shut-Valve-with-Loss
;;;          ---
;;;         __|__
;;;       ==|_|_|==
;;;  When valve is open, flow is linearly proportional to pressure
;;;  difference.  The difference of the pressure is loss.
;;;  Fluid flows to both directions.
;;;-------------------------------------------------------------------

(define-component Shut-Valve-with-Loss hydraulic
  "Shut Valve with Loss"
  (terminal-variables (port1 (Flow1 flow)
                             (P1    pressure))
                      (port2 (Flow2 flow)
                             (P2    pressure)))
  (component-variables (PDiff     pressure)
                       (FlowCoef  void (quantity-space (0 FC inf))))
  (mode-variables (mode open closed))
  (constraints ((add P2 PDiff P1))
               ((constant FlowCoef FC))
               ((mode open)
                -> ((minus Flow1 Flow2))
                   ((mult PDiff FlowCoef Flow1)))
               ((mode closed)
                -> ((zero-std Flow1))
                   ((zero-std Flow2)))))

;;;-------------------------------------------------------------------
;;;  CC-component:  Regulator
;;;             +----+
;;;           __|__  |
;;;           |_|_|  |
;;;         ==|___|====
;;;  Fluid flows from inlet to outlet (no reversal flow occurs)
;;;  Reg-model: closed -- No flow forever.
;;;  Reg-mode:  open -- failed regulator with open valve (free flow).
;;;             closed -- failed regulator with closed valve (no flow).
;;;-------------------------------------------------------------------

(define-component Regulator hydraulic
  "Regulator"
  (terminal-variables (inlet  (Flow-i flow)
                              (P-i    pressure))
                      (outlet (Flow-o flow)
                              (P-o    pressure
                                      (quantity-space (0 P-reg inf)))))
  (component-variables (PDiff pressure))
  (mode-variables
    (reg-model (invisible
                <- (and (PDiff (0 dec))
                         (not (Flow-i (0 nil))))
                   :eval (format t "~%-> Too Low Regulator Pressure"))
               (pipe
                <- (and (P-o (P-reg dec))
                        (reg-model (ideal-psource std))
```

```
                        (reg-mode  (regulated std)))
                   :eval (format t "~%-> Regulator becomes a pipe"))
              (closed
                <- (and (PDiff (0 dec)) (Flow-i (0 nil))
                        (not (reg-model (closed std))))
                   :eval (format t "~%-> Regulator is closed"))
              (ideal-psource
                <- (and (P-o (P-reg inc)) (reg-model (pipe std))
                        (reg-mode  (regulated std)))
                   :eval (format t "~%-> Ideal Pressure Source")
                   :assert ((P-o (P-reg std)))))
                (reg-mode open regulated closed))
  (constraints ((add PDiff P-o P-i))
              ((minus Flow-i Flow-o))
              ((or (reg-model closed) (reg-mode closed))
               -> ((zero-std Flow-i)))
              ((and (or (reg-model pipe) (reg-mode open))
                    (not (reg-mode closed)))
               -> ((m+ PDiff Flow-i) (0 0) (inf inf)))
              ((and (reg-model ideal-psource) (reg-mode regulated))
               -> ((constant P-o P-reg)))
              ((and (reg-model invisible)
                    (not (reg-mode closed)))
               -> ((zero-std PDiff)))))

;;;------------------------------------------------------------------
;;;  CC-component:  Gas+Liquid-Tank
;;;  Gas pushes out liquid, or liquid pushes out gas.
;;;                  +-------------------+
;;;                  |        #          |
;;;          PortG ==|   Gas  #  Liquid  |==   PortL
;;;                  |        #          |
;;;                  +-------------------+
;;;  Gas is governed by the equation of state.
;;;  While Gas does not leak, Liquid may leak.
;;;  The pressure in a tank is uniform.
;;;  There are three levels of the amounts of gas and liquid, and the
;;;   volumes of them.  They are "init", "max" and "nom".
;;;------------------------------------------------------------------

(define-component Gas+Liquid-Tank hydraulic
  "Gas and Liquid Mixture in a Tank"
  (terminal-variables (portG (FlowG flow)
                             (PG    pressure))
                      (portL (FlowL flow)
                             (PL    pressure)))
  (component-variables (AmtGas amount
                               (quantity-space
                                 (0 AmtGInit AmtGNom AmtGmax)))
                       (VolGas void
                               (quantity-space (0 VGInit VGNom VolTotal)))
                       (nRT void
                            (quantity-space (0 nRTInit nRTNom nRTmax)))
                       (AmtLiquid amount
                                  (quantity-space
```

```
                                    (0 AmtLNom AmtLInit AmtLmax)))
                      (dAmtLiquid void)
                      (VolLiquid void
                                 (quantity-space
                                    (0 VLNom VLInit VolTotal)))
                      (DenLiquid void (quantity-space (0 DLmax)))
                      (P pressure (quantity-space (0 Pmax)))
                      (VolTotal void (quantity-space (voltotal)))
                      (PDiffEnv pressure)              ;; P - Penv
                      (LeakCoef void (quantity-space (0 LC inf)))
                      (LeakFlow flow))
  (mode-variables (Leak? yes no))
  (components (Environment Space-Environment))
  (constraints ((d/dt AmtGas FlowG))
               ((d/dt AmtLiquid dAmtLiquid))
               ((equal P PG))
               ((equal P PL))
               ((add PDiffEnv (Environment P) P))
               ((Leak? yes) -> ((constant LeakCoef LC)))
               ((Leak? no) -> ((zero-std LeakCoef)))
               ((mult LeakCoef PDiffEnv LeakFlow))
               ((add LeakFlow dAmtLiquid FlowL))
               ((m+ AmtGas nRT) (0 0) (AmtGInit nRTInit)
                                (AmtGNom nRTNom) (AmtGmax nRTmax))
               ((mult P VolGas nRT) (0 0 0) (Pmax VolTotal nRTmax))
               ((m+ P DenLiquid) (0 0) (Pmax DLmax))
               ((mult DenLiquid VolLiquid AmtLiquid))
               ((add VolGas VolLiquid VolTotal) (VolTotal 0 VolTotal)
                                                (0 VolTotal VolTotal)
                                                (VGNom VLNom VolTotal)
                                                (VGInit VLInit VolTotal))
               ((constant VolTotal))))

;;;-----------------------------------------------------------------
;;;  CC-component:  Pipe-with-no-Leak-and-no-Loss
;;;           -----------------------------------
;;;           -----------------------------------
;;;  Incompressible fluid flows into both directions.
;;;  There is no loss.  This, therefore, implies P1 = P2.
;;;-----------------------------------------------------------------

(define-component Pipe-with-no-Leak-and-no-Loss hydraulic
  "Pipe with loss and leak"
  (terminal-variables (port1   (Flow1 flow)
                               (P1    pressure))
                      (port2   (Flow2 flow)
                               (P2    pressure)))
  (constraints ((equal P1 P2))
               ((minus Flow1 FLow2))))

;;;-----------------------------------------------------------------
;;;  CC-component:  Pipe-with-Leak-and-no-Loss
;;;           -----------------------------------
;;;           -----------------------------------
;;;  Compressible fluid flows into both directions.
```

```
;;;  The amount of leak is linearly proportional to the difference of
;;;   the average pressure in pipe and the pressure of the space.
;;;  There is no loss.  This, therefore, implies P1 = P2.
;;;-------------------------------------------------------------------

(define-component Pipe-with-Leak-and-no-Loss hydraulic
  "Pipe with loss and leak"
  (terminal-variables (port1 (Flow1 flow)
                             (P1    pressure))
                      (port2 (Flow2 flow)
                             (P2    pressure)))
  (component-variables (Amt          amount (quantity-space (0 Amtmax)))
                       (dAmt         flow)
                       (P            pressure)
                       (PDiffEnv     pressure)             ;; P - Penv
                       (LeakCoef     void (quantity-space (0 LC inf)))
                       (LeakFlow     flow)                 ;; Leaking flow
                       (SumFlow      flow))                ;; Flow1 + Flow2
  (mode-variables (Leak? yes no))
  (components (Space Space-Environment))
  (constraints ((d/dt Amt dAmt))
               ((m+ Amt P) (0 0) (Amtmax Pmax))
               ((equal P P1))
               ((equal P P2))
               ((add (Space P) PDiffEnv P))
               ((mult PDiffEnv LeakCoef LeakFlow))
               ((add Flow1 Flow2 SumFlow))
               ((add LeakFlow dAmt SumFlow))
               ((Leak? yes) -> ((constant LeakCoef LC)))
               ((Leak? no)  -> ((zero-std LeakCoef)))))

;;;-------------------------------------------------------------------
;;;  CC-component:  2-Port-Tank-with-Leak
;;;          +----------+
;;;          |          |
;;;        ==|          |==
;;;          |          |
;;;          +----------+
;;;  Compressible fluid flows into both directions.
;;;  Pressure is uniform in tank.
;;;  The amount of leak is linearly proportional to the difference of
;;;  the pressure Leak flow is from inside to outside of tank.
;;;-------------------------------------------------------------------


(define-component 2-Port-Tank-with-Leak hydraulic
  "Manifold"
  (terminal-variables (port1 (Flow1  flow)
                             (P1     pressure))
                      (port2 (Flow2  flow)
                             (P2     pressure)))
  (component-variables (Amt amount (quantity-space (0 Amtmax)))
                       (dAmt void)
                       (P    pressure  (quantity-space (0 Pmax)))
                       (LeakFlow flow)
```

```
                      (LeakCoef void (quantity-space (0 LC inf)))
                      (PDiffEnv pressure)
                      (SumFlow flow))                ;; Flow1 + Flow2
  (mode-variables (Leak? yes no))
  (components (Environment Space-Environment))
  (constraints ((d/dt Amt dAmt))
               ((m+ Amt P) (0 0) (Amtmax Pmax))
               ((equal P P1))
               ((equal P P2))
               ((add PDiffEnv (Environment P) P))
               ((mult LeakCoef PDiffEnv LeakFlow))
               ((add Flow1 Flow2 SumFlow))
               ((add LeakFlow dAmt SumFlow))
               ((Leak? yes)
                -> ((constant LeakCoef LC)))
               ((Leak? no)
                -> ((zero-std LeakCoef)))))

;;;----------------------------------------------------------------------
;;;  CC-component:  Thruster
;;;            /
;;;         __/
;;;         --
;;;           \
;;;            \
;;;  Thruster of rocket engine.
;;;----------------------------------------------------------------------

(define-component Thruster hydraulic
  "Thruster"
  (terminal-variables (port (Flow flow)
                            (P    pressure (quantity-space (0 Pmax)))))
  (component-variables (PDiff pressure)
                      (DenLiquid    void (quantity-space (0 inf)))
                      (Den*PDiff    void (quantity-space (0 inf)))
                      (f[Den*PDiff] void (quantity-space (0 inf)))
                      (Thruster     void (quantity-space (0 max))))
  (components (Space Space-Environment))
  (constraints ((add PDiff (Space P) P))
               ((mult PDiff DenLiquid Den*PDiff))
               ((m+ P DenLiquid) (0 0) (Pmax inf))
               ((m+ Den*PDiff f[Den*PDiff]) (0 0) (inf inf))
               ((constant Thruster))
               ((mult Thruster f[Den*PDiff] Flow))
               ((w+ PDiff Den*PDiff))))
```

B RCS Models

B.1 Four Components RCS Models

```
;;;----------------------------------------------------------------------
;;;  CC-component:  RCS-component#1  (Four Components Model)
;;;     CC-component:  He-Tank    <--  1-Port-Tank-with-Leak#1
;;;     CC-component:  Regulator  <--  Abstracted-Regulator
;;;     CC-component:  Fuel-Tank  <--  Ullage-Liquid-Tank-with-Line
;;;     CC-component:  Manifold+Thruster  <--  Manifold-with-Thruster
;;;----------------------------------------------------------------------

(define-component RCS-component#1 hydraulic
  "RCS component"
  (components (He-Tank
                1-Port-Tank-with-Leak#1
                (quantity-spaces
                  (Flow (minf 0))
                  (P    (0 PHeMin PHePreg PHeSreg PHeInit))
                  (Amt  (0 AmtHeMin AmtHePreg AmtHeSreg AmtHeInit))
                  (TotalLeak (0 inf))
                  (Outflow (0 inf))
                  (MFlow (0 inf))))
              (Regulator
                Abstracted-Regulator)
              (Fuel-Tank
                Ullage-Liquid-Tank-with-Line
                (quantity-spaces
                  (InletP   (0 PUllMin PUllPreg PUllSreg PUllMax))
                 (DenLiquid (0 DLPreg DLMax))))
              (Manifold+Thruster
                Manifold-with-Thruster))
  (connections (He-Reg   (He-Tank port) (Regulator inlet))
               (Reg-Fuel (Regulator outlet) (Fuel-Tank inlet-side))
               (Fuel-Man (Fuel-Tank outlet-side) (Manifold+Thruster port)))
  (mode-variables
    (operation initial-state
               (:stop
                 <- (or ((Fuel-Tank Flow-i) (0 dec))
                        ((Fuel-Tank InletP) (PUllMin dec))))
               (reg-set-low-pressure
                 <- (and ((Regulator PDiff) (0 dec))
                         (not ((Regulator Flow-i) (0 nil))))
                    :eval (format t "~%-> reg-set-low-pressure")
                    :assert (((Regulator reg-model) (invisible std)))
                    :inherit-qmag :rest
                    :inherit-qdir ((He-Tank P) (Manifold+Thruster P)
                                   (Fuel-Tank TankValve)
                                   (Fuel-Tank VolTotal)))
               (Sreg-pipe
                 <- (and ((Fuel-Tank InletP) (PUllPreg nil))
                         ((Regulator reg-model) (ideal-psource std))
                         ((Regulator reg-mode) (sreg std)))
                    :eval (format t "~%-> Sreg-pipe")
                    :assert (((Regulator reg-model) (pipe std)))
```

```
     :inherit-qmag
     (test-function
       #'(lambda (varname)
           (if (member varname
                       '((cc-name (He-Tank Flow))
                         (cc-name (He-Tank Mflow))
                         (cc-name (Regulator Flow-i))
                         (cc-name (Regulator Flow-o))
                         (cc-name (Fuel-Tank Flow-i))
                         (cc-name (He-Tank Outflow))
                         (cc-name (He-Tank dAmt))
                         (cc-name (Regulator reg-model))))
               nil
               t)))
     :inherit-qdir ((Manifold+Thruster P)
                    (Fuel-Tank TankValve)
                    (Fuel-Tank VolTotal)))
(check-valve-closed
  <- (and ((Regulator PDiff) (0 dec))
          ((Regulator Flow-i) (0 nil))
          (not ((Regulator reg-model) (closed std))))
     :eval (format t "~%-> check-valve-closed")
     :assert (((Regulator reg-model) (closed std)))
     :inherit-qmag :rest
     :inherit-qdir :rest)
(ideal-psource-preg
  <- (and ((Fuel-Tank InletP) (PUllPreg inc))
          ((Regulator reg-model) (pipe std))
          ((Regulator reg-mode) (preg std)))
     :eval (format t "~%-> ideal-psource-preg")
     :assert (((Regulator reg-model) (ideal-psource std))
              ((Fuel-Tank InletP) (PUllPreg std)))
     :inherit-qmag
     (test-function
       #'(lambda (varname)
           (if (member varname
                       '((cc-name (He-Tank Flow))
                         (cc-name (He-Tank Mflow))
                         (cc-name (Regulator Flow-i))
                         (cc-name (Regulator Flow-o))
                         (cc-name (Fuel-Tank Flow-i))
                         (cc-name (He-Tank Outflow))
                         (cc-name (He-Tank dAmt))))
               nil
               t)))
     :inherit-qdir ((Fuel-Tank TankValve)
                    (Fuel-Tank VolTotal)))
(ideal-psource-sreg
  <- (and ((Fuel-Tank InletP) (PUllSreg inc))
           ((Regulator reg-model) (pipe std))
           ((Regulator reg-mode) (sreg std)))
     :eval (format t "~%-> ideal-psource-sreg")
     :assert (((Regulator reg-model) (ideal-psource std))
              ((Fuel-Tank InletP) (PUllSreg std)))
     :inherit-qmag
```

```
                (test-function
                  #'(lambda (varname)
                      (if (member varname
                                  '((cc-name (He-Tank Flow))
                                    (cc-name (He-Tank Mflow))
                                    (cc-name (Regulator Flow-i))
                                    (cc-name (Regulator Flow-o))
                                    (cc-name (Fuel-Tank Flow-i))
                                    (cc-name (He-Tank Outflow))
                                    (cc-name (He-Tank dAmt))))
                          nil
                          t)))
                :inherit-qdir ((Fuel-Tank TankValve)
                               (Fuel-Tank VolTotal)))))
  (constraints
    ((m+ (He-Tank Amt) (He-Tank P))
     (0 0) (AmtHeMin PHeMin) (AmtHePreg PHePreg) (AmtHeSreg PHeSreg)
     (AmtHeInit PHeInit))
    ((add (Regulator PDiff) (Fuel-Tank InletP) (He-Tank P))
     (0 PUllMin PHeMin) (0 PUllPreg PHePreg) (0 PUllSreg PHeSreg))
    ((add (Fuel-Tank PDiff) (Manifold+Thruster P) (Fuel-Tank InletP))
     (0 Ppreg PUllPreg))
    ((mult (Fuel-Tank InletP) (Fuel-Tank VolUll) (Fuel-Tank nRT[AmtUll]))
     (0 0 0) (PUllPreg VUInit nRTInit) (PUllPreg VUNom nRTNom)
     (PUllMax VolTotal nRTMax))
    ((m+ (Fuel-Tank InletP) (Fuel-Tank DenLiquid))
     (0 0) (PUllPreg DLPreg) (PUllMax DLMax))
    ((mult (Fuel-Tank DenLiquid)
           (Fuel-Tank VolLiquid)
           (Fuel-Tank AmtLiquid))
     (DLPreg VLInit AmtLInit) (DLPreg VLNom AmtLNom))
    ((or ((Regulator reg-model) ideal-psource)
         ((Regulator reg-model) invisible))
     -> ((m+ (Fuel-Tank LiquidOutFlow) (Fuel-Tank Flow-i))
         (0 0) (inf inf)))
    (((Regulator reg-model) invisible)
     -> ((m+ (He-Tank P) (Fuel-Tank InletP))
         (0 0) (PHeMin PUllMin) (PHePreg PULLPreg) (PHeSreg PULLSreg)
         (PHeInit PUllMax)))
    ((and ((Regulator reg-model) pipe) ((Fuel-Tank TotalLeak) 0)
          ((Fuel-Tank ValveFlowCoef) max) ((Manifold+Thruster Thruster) 0)
          ((Manifold+Thruster TotalLeak) 0))
     -> ((w+ (Fuel-Tank InletP) (Fuel-Tank nRT[AmtUll]))))))
```

B.2 Ten Components RCS Models

```
;;;-----------------------------------------------------------------
;;;  CC-component:  RCS-component#2  (Ten Components Model)
;;;     CC-component:  He-Tank      <--  1-Port-Tank-with-leak#2
;;;     CC-component:  He-Valve     <--  Shut-valve-without-loss
;;;     CC-component:  S-Regulator  <--  Regulator
;;;     CC-component:  P-Regulator  <--  Regulator
;;;     CC-component:  Fuel-Tank    <--  Gas+liquid-tank
```

```
;;;     CC-component:  Tank-Valve   <--  Shut-valve-with-loss
;;;     CC-component:  Fuel-Line    <--  Line-with-Leak-and-no-Loss
;;;     CC-component:  Man-Valve    <--  Shut-valve-without-loss
;;;     CC-component:  Manifold     <--  2-Port-Tank-with-Leak
;;;     CC-component:  Thruster     <--  Thruster
;;;-----------------------------------------------------------------

(define-component RCS-component#2 hydraulic
  "RCS component"
  (components (He-Tank 1-Port-Tank-with-Leak#2
                       (quantity-spaces
                         (Flow (minf 0))
                         (P    (0 PHeMin PHePreg PHeSreg PHeInit inf))
                         (Amt  (0 AmtHeMin AmtHePreg AmtHeSreg AmtHeInit))
                         (LeakFlow (0 inf))))
              (He-Valve Shut-Valve-without-Loss
                         (quantity-spaces (P2 (0 PPreg PSreg PInit inf))))
              (S-Regulator Regulator
                            (quantity-spaces (P-o (0 PPreg P-reg inf))))
              (P-Regulator Regulator
                            (quantity-spaces (P-o (0 P-reg PSreg inf))))
              (Fuel-Tank   Gas+Liquid-Tank
                            (quantity-spaces (P (0 PMin PPreg PSreg PMax inf))
                                             (DenLiquid (0 DLPreg DLMax))))
              (Tank-Valve  Shut-Valve-with-Loss)
              (Fuel-Line   Pipe-with-Leak-and-no-Loss
                            (quantity-spaces (P   (0 PPreg PMax))
                                             (Amt (0 AmtPreg AmtMax))))
              (Man-Valve   Shut-Valve-without-Loss)
              (Manifold    2-Port-Tank-with-Leak
                            (quantity-spaces (P (0 PPreg PMax))
                                             (Amt (0 AmtPreg AmtMax))))
              (Thruster    Thruster
                            (quantity-spaces (P (0 PPreg PMax)))))
  (connections (He-Valve      (He-Tank port)        (He-Valve port1))
               (Valve-SReg    (He-Valve port2)      (S-Regulator inlet))
               (SReg-PReg     (S-Regulator outlet)  (P-Regulator inlet))
               (PReg-Fuel     (P-Regulator outlet)  (Fuel-Tank portG))
               (Fuel-Valve    (Fuel-Tank portL)     (Tank-Valve port1))
               (Valve-Line    (Tank-Valve port2)    (Fuel-Line port1))
               (Line-Valve    (Fuel-Line port2)     (Man-Valve port1))
               (Valve-Man     (Man-Valve port2)     (Manifold port1))
               (Man-Thruster  (Manifold port2)      (Thruster port)))
  (mode-variables (operation
                   initial-state
                   (:stop
                    <- (or ((Fuel-Tank FlowG) (0 dec))
                           ((Fuel-Tank P) (PMin dec))))
                   (Regulator-low-pressure
                    <- (or (and ((P-Regulator PDiff) (0 dec))
                                (not ((P-Regulator Flow-i) (0 nil))))
                           (and ((S-Regulator PDiff) (0 dec))
                                (not ((S-Regulator Flow-i) (0 nil)))))
                    :inherit-qmag :rest
                    :eval (format t "~%What a low pressure is!")
```

```
 :inherit-qdir ((He-Tank P) (Manifold P)
               (Tank-Valve mode) (Fuel-Tank VolTotal)))
(Regulator-Pipe-Flow
 <- (or (and ((Fuel-Tank P) (PPreg dec))
             ((P-Regulator reg-model) (ideal-psource std))
             ((P-Regulator reg-mode) (regulated std)))
        (and ((S-Regulator P-o) (P-reg dec))
             ((S-Regulator reg-model) (ideal-psource std))
             ((S-Regulator reg-mode) (regulated std))))
 :inherit-qmag
 (test-function
  #'(lambda (varname)
      (if (member
           varname
           '((cc-name (He-Tank Flow))
             (cc-name (He-Tank dAmt))
             (cc-name (He-Valve Flow1))
             (cc-name (He-Valve Flow2))
             (cc-name (S-Regulator Flow-i))
             (cc-name (S-Regulator Flow-o))
             (cc-name (S-Regulator reg-model))
             (cc-name (P-Regulator Flow-i))
             (cc-name (P-Regulator Flow-o))
             (cc-name (P-Regulator reg-model))
             (cc-name (Fuel-Tank FlowG))))
          nil
          t)))
 :inherit-qdir ((Manifold P) (Tank-Valve mode)
               (Fuel-Tank VolTotal)))
(check-valve-closed
 <- (or (and ((P-Regulator PDiff) (0 dec))
             ((P-Regulator Flow-i) (0 nil))
             (not ((P-Regulator reg-model) (closed std))))
        (and ((S-Regulator PDiff) (0 dec))
             ((S-Regulator Flow-i) (0 nil))
             (not ((S-Regulator reg-model) (closed std)))))
 :inherit-qmag :rest
 :inherit-qdir :rest)
(ideal-psource
 <- (or (and ((P-Regulator P-o) (P-reg inc))
             ((P-Regulator reg-model) (pipe std))
             ((P-regulator reg-mode) (regulated std)))
        (and ((S-Regulator P-o) (P-reg inc))
             ((S-Regulator reg-model) (pipe std))
             ((S-regulator reg-mode) (regulated std))))
 :inherit-qmag
 (test-function
  #'(lambda (varname)
      (if (member
           varname
           '((cc-name (He-Tank Flow))
             (cc-name (He-Tank dAmt))
             (cc-name (He-Valve Flow1))
             (cc-name (He-Valve Flow2))
             (cc-name (S-Regulator Flow-i))
```

```
                              (cc-name (S-Regulator Flow-o))
                              (cc-name (P-Regulator Flow-i))
                              (cc-name (P-Regulator Flow-o))
                              (cc-name (Fuel-Tank FlowG))))
                           nil
                           t)))
                  :inherit-qdir ((Tank-Valve mode) (Fuel-Tank VolTotal)))))
(qspace-hierarchy (((He-Tank P) (He-Valve P2) (S-Regulator P-o)
                    (P-Regulator P-o) (Fuel-Tank P))
                   ->
                   (*seq 0
                         (*eqv (He-Tank P PHeMin) (Fuel-Tank P Pmin))
                         (*eqv (He-Tank P PHePreg) (He-Valve P2 PPreg)
                               (S-Regulator P-o PPreg)
                               (P-Regulator P-o P-reg)
                               (Fuel-Tank P PPreg))
                         (*eqv (He-Tank P PHeSreg) (He-Valve P2 PSreg)
                               (S-Regulator P-o P-reg)
                               (P-Regulator P-o PSreg)
                               (Fuel-Tank P PSreg))
                         (*eqv (He-Tank P PHeInit) (He-Valve P2 PInit))
                         inf)))
(constraints ((m+ (He-Tank Amt) (He-Tank P))
              (0 0) (AmtHeMin PHeMin) (AmtHePreg PHePreg) (AmtHeSreg PHeSreg)
              (AmtHeInit PHeInit))
             (((He-Valve mode) open)
             -> ((= (He-Tank P) (He-Valve P2))))
             ((add (S-Regulator PDiff) (S-Regulator P-o) (He-Valve P2))
              (0 P-reg PSreg))
             ((and (not ((S-Regulator Reg-mode) closed))
                   ((S-Regulator Reg-model) invisible))
              -> ((= (He-Valve P2) (S-Regulator P-o))))
             ((add (P-Regulator PDiff) (P-Regulator P-o) (S-Regulator P-o))
              (0 P-reg PPreg) (0 PSreg P-reg))
             ((= (P-Regulator P-o) (Fuel-Tank P)))
             ((and (not ((P-Regulator Reg-mode) closed))
                   ((P-Regulator Reg-model) invisible))
              -> ((= (S-Regulator P-o) (Fuel-Tank P))))
             ((mult (Fuel-Tank P) (Fuel-Tank VolGas) (Fuel-Tank nRT))
              (0 0 0) (PPreg VGInit nRTInit) (PPreg VGNom nRTNom)
              (PMax VolTotal nRTMax))
             ((m+ (Fuel-Tank P) (Fuel-Tank DenLiquid))
              (0 0) (PPreg DLPreg) (PMax DLMax))
             ((mult (Fuel-Tank DenLiquid)
                    (Fuel-Tank VolLiquid)
                    (Fuel-Tank AmtLiquid))
              (DLPreg VLInit AmtLInit) (DLPreg VLNom AmtLNom))
             ((and (or ((S-Regulator Reg-model) ideal-psource)
                       ((S-Regulator Reg-model) invisible))
                   (or ((P-Regulator Reg-model) ideal-psource)
                       ((P-Regulator Reg-model) invisible)))
              -> ((m- (Fuel-Tank FLowG) (Fuel-Tank FLowL))
                  (minf inf) (0 0) (inf minf)))
             ((and ((He-Valve mode) open)
                   ((S-Regulator Reg-model) invisible)
```

```
      (not ((S-Regulator Reg-mode) open))
      ((P-Regulator Reg-model) invisible)
      (not ((P-Regulator Reg-mode) open)))
 -> ((m+ (He-Tank p) (Fuel-Tank P))
     (0 0) (PHeMin PMin) (PHePreg PPreg) (PHeSreg PSreg)
     (PHeInit PMax)))
((and ((S-Regulator Reg-model) invisible)
      (not ((S-Regulator Reg-mode) closed))
      (or ((P-Regulator Reg-model) pipe)
          ((P-Regulator Reg-model) invisible))
      (not ((P-Regulator Reg-mode) closed))
      ((Fuel-Tank Leak?) no)
      ((Fuel-Line Leak?) no)
      ((Manifold Leak?) no)
      ((Thruster Thruster) 0))
 -> ((w+ (Fuel-Tank P) (Fuel-Tank nRT))))
(((Tank-Valve mode) open)
 -> ((add (Tank-Valve PDiff) (Fuel-Line P) (Fuel-Tank P))
     (0 PPreg PPreg)))
((m+ (Fuel-Line Amt) (Fuel-Line P))
 (0 0) (AmtPreg PPreg) (AmtMax PMax))
(((Man-Valve mode) open)
 -> ((equal (Fuel-Line P) (Manifold P))
     (PPreg PPreg) (PMax PMax)))
((m+ (Manifold Amt) (Manifold P))
 (0 0) (AmtPreg PPreg) (AmtMax PMax))
((equal (Manifold P) (Thruster P)) (PPreg PPreg) (PMax PMax))))
```