A New Fault Injection Method for Liquid Rocket Pressurization and Feed Systems

Zhu Mingqing[1] Xie Gang[1] Shao Jintao[2] Chen Liping[1] Zhou Fanli[2]

[1]CAD Centre, Huazhong University of Science and Technology, Wuhan, China, 430074
[2]Suzhou Tongyuan Software&Control Tech. Co., Suzhou, China, 215123
{zhumq,xieg,shaojt,chenlp,zhoufl}@tongyuan.cc

Abstract

Fault simulation is an important method in the design of liquid rockets and fault injection is necessary for fault simulation. In this paper, we present a new fault injection method for liquid rocket pressurization and feed systems (PFS) without modifying the system structure. Firstly, we develop a physics-based model of pressurization and feed systems based on Modelica, which describes both nominal and faulty behaviors in a unified way. Then, we describe the new fault injection method, which uses the fault mode library and constructs the association between the Modelica model and the fault mode using customized Modelica annotation in MWorks®. We verify the new method by simulating several typical fault modes such as leakage and clogging. The results show that our method could be easily used to simulate various fault modes in liquid rocket pressurization and feed systems. Moreover, the new fault simulation process indeed plays a role in the system design. Our results could provide some reference for the ongoing research in fault detection and diagnoses.

Keywords: fault simulation; fault injection; Modelica/MWorks; pressurization; fault mode

1 Introduction

A suitable pressurization and feed system is important for a liquid rocket to transfer rocket propellants from the propellant tanks to the engine at certain flow rates and pressures (Partola, 2012). Mostly, the engine could not generate enough thrust to keep the rocket in orbit if the pressure inside the propellant tank is too low, which may lead to flight failure. Therefore, a diagnostic solution is needed to quickly identify the faults so that recovery actions can be taken or an abort procedure can be initiated before system safety is compromised (Daigle, 2011). Effective diagnoses require abundant historic data, which are traditionally acquired by executing many ground tests. Unfortunately, the costs of implementing ground tests are enormous and the process of physical tests is extremely dangerous. In addition, it is really hard or impossible to reappear some fault modes because of equipment restrictions, let alone covering all possible flight conditions. Nowadays, with the development of computer science, fault simulation based on numerical methods is an ideal alternative to ground tests. Relying on a detailed model of system behaviors under nominal and faulty conditions (Daigle, 2011), numerical simulation has several advantages. Firstly, an engineer could build a mathematical PFS model according to its physical function and simulate its behaviors under all kinds of working conditions to verify the design of the system. Moreover, typical fault modes of a system could be manually injected into the nominal model to predict their effects on the system performance. In this way, we could accumulate abundant faulty knowledge of the system, which is important for ongoing research in fault diagnoses and design optimization.

Implementing faults in PFS is not new to the literature. For example, Gao Ming et constructed a filling system based on Modelica where fault simulation was processed by altering the models or resetting the parameters (Gao, 2009).

Wang Min et built a modular PFS based on Matlab where two typical faults were simulated by adding step signals to change the behavior of the system (Wang, 2010).

Fan Zhongze et simulated four kinds of faults by VC++ where the fault is expressed by the fault factor and fault trigger time (Fan, 2008).

Another approach for model-based fault simulation is used by F.L.J. van der Linden. Using instance modifiers as well as an inner-outer broadcasting method, the faults can be triggered in a central block (F.L.J. van der Linden, 2014). Though valuable in some aspects, this method is hard to handle a large number of fault modes.

Most of these works are examples in which faults in PFS are triggered and simulated. The simulation results can give some reference to the model-based diagnoses. However, all the implementations have to modify the original system model to inject fault modes, which is very fussy if there are many fault modes. In fact, engineers prefer a simpler and more convenient way to inject faults in system models.

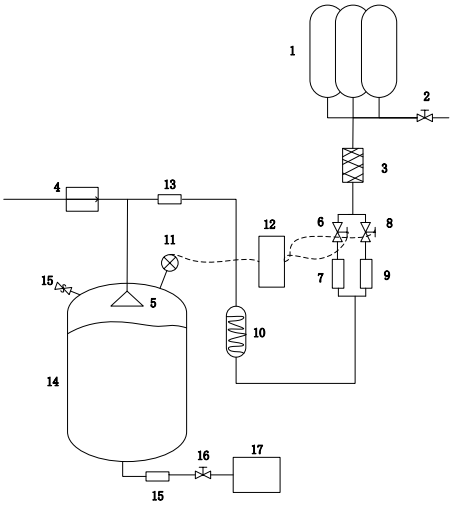
In this paper, we develop a lumped-parameter dynamic model of PFS using Modelica on MWorks®, which is simple enough to allow for physics analyses and numerical simulations. More importantly, we propose a new method to describe the fault mode and inject the fault mode into the Modelica model without modifying the structure of the original system. Several typical fault modes are simulated to justify the validity of this method. We have investigated both the nominal regime and the effects of several primary faults, such as gas leakage in the gas pipe as well as clogging of the electric valve.

The paper is organized as follows. Section 2 gives an introduction to liquid rocket pressurization and feed system. Section 3 briefly describes the modeling of PFS considering both nominal and faulty behaviors. Section 4 presents the new method of model-based fault injection and the workflow of fault simulation. Section 5 verifies the approach with several fault simulation experiments. Conclusions and suggestions for future work are presented in section 6.

2 Overview of PFS

A simplified working diagram of the liquid rocket PFS is depicted in Figure 1.

The working process is outlined as follows. Initially, cold helium gas is injected into gas bottles 1 through valve 2 to obtain very high pressure on the ground. At the same time, the propellant tank is pressurized through ground valve 4. Once engine 17 starts, high pressure gas is released from the gas bottle, and finally enter the propellant tank. The mass flow rate of the gas is controlled by controller 12, which controls the opening of SV 6 or SV 8 by detecting the pressure in the propellant tank.



1-Gas Bottles; 2-Injection Valve; 3-Filter
4-Ground Valve; 5-Baffle; 6-Main SV
7-Main Board; 8-Second SV; 9-Second Board
10-Heater; 11-Pressure Sensor; 12-Controller
13-Gas Pipe； 14-Oxidant Tank; 15-Transport Pipe
16-Pump Valve; 17-Engine

Figure 1 A Liquid Rocket PFS

3 Modeling PFS

A detailed system model is essential for fault simulation. Based on the above analysis, fluid dynamics and hydraulics, we could construct all the nominal mathematical models. To meet the requirements of both validity and computational performance, all the models are treated lumped-parameter. Both nominal and faulty behaviors are considered in the model.

Pressure Board

The purpose of the pressure board is to control the flow rate of the fluid from upwind. In fact, the board could be considered as an orifice. Based on the continuous equation, the isentropic relationship and the equation of the gas velocity, we could obtain the flow rate of the gas according to the pressure ratio (Esposito, 2000).

The mass flow rate is given by:

$$\dot{m} = A \bullet C_q \bullet C_m \frac{P_{up}}{\sqrt{T_{up}}} \tag{1}$$

Where Cq is the discharge coefficient, A is the restriction area, P_{up} is the upstream pressure, C_m is the mass flow parameter, and T_{up} is the upstream temperature.

The critical pressure ratio at which the flow switches from sonic to subsonic can be calculated using the equation:

$$P_{cr} = \left(\frac{2\gamma_s}{\gamma_s + 1} \right)^{\frac{1}{1-\gamma_s}} \tag{2}$$

Here, γ_s is the isentropic calorific factor.

When $(P_{dn} / P_{up} > P_{cr})$, the flow is subsonic and the flow parameter C_m is a function of the pressure ratio and depends on the gas properties. When $(P_{dn} / P_{up} > P_{cr})$, the flow is sonic, the mass flow parameter C_m is a constant and the mass flow rate only depends on the upstream temperature.

The nominal behavior of the board is described as in equation (1). To characterize the fault mode such as leakage, we introduce a leakage variable *dm_flow* and leakage coefficient K (0<=K<=1). The constrain equation is:

$$dm\_flow = K \cdot \dot{m}_{g\_in} \tag{3}$$

Here, K is the fault parameter of the board. K=0 implies the nominal condition and K=1 implies complete leakage.

Pipes (Gas and Liquid)

The flow condition of the fluid could be treated as laminar if the flow rate is relatively small and the mass flow rate changes linearly with the pressure drop (Esposito, 2000). The equation is given by:

DOI 10.3384/ecp15118557

$$\dot{m} = G \cdot dp \tag{4}$$

Similarly, we introduce a fault parameter *leak* to represent the leakage fault mode. The equation is given by:

$$dm\_flow = leak \cdot \dot{m} \tag{5}$$

Here, *leak*=0 implies the nominal condition and *leak*=1 implies total leakage.

Gas Bottle

Gas Bottle is a closed volume filled with inert gas. Before the flight of the rocket, the gas bottle will be injected into enough gas to obtain high pressure. When the valve opens, the gas quickly expanses into the downstream road. Since the gas flow is very fast, the heat transfer between the wall and the gas could be neglected and the temperature and pressure in the gas bottle would be given by (Esposito, 2000):

$$T = T_0 \left(\frac{p}{p_0} \right)^{\frac{\kappa-1}{\kappa}} \tag{6}$$

$$\frac{dp}{dt} = \frac{\kappa \cdot R \cdot T}{V} \dot{m} \tag{7}$$

Here, T_0 is the initial temperature in the gas bottle, p_0 is the initial pressure.

We introduce coefficient *leak* to represent the leakage behavior of the bottle. The equation is:

$$dm\_flow = leak \cdot \dot{m} \tag{8}$$

Propellant Tank

The propellant tank is the most important component of a PFS. Tank pressurization is a very complicated physical process: the outer gas enters the tank and mixes with the initial gas in the tank, following the process of heat transfer and mass transfer. The tank wall is surrounded with the environment with all kinds of heat radiation and convective heat flow. Hence, it is rather difficult to predict the pressure in the tank. To determine the mathematical model, several assumptions are made as follows:

- The gas is undissolved with the liquid.
- The liquid is considered incompressible.
- The phenomena such as condensation and boiling are neglected.

Under low pressure, the gas could be treated as the ideal gas. The equation of state is given by:

$$pV = mRT \tag{9}$$

Ignore the kinetic energy of the entering gas. The mixed gas transfers heat with the tank wall and the liquid. The conversation equation is given by (Esposito, 2000):

$$\frac{dU_{gas}}{dt} = \sum \dot{m}_{in} \cdot c_p \cdot T_{in} - \sum (Q_1 + Q_2) - p_{gas} \bullet der(V_{gas}) \tag{10}$$

Here, U_{gas} is the internal energy of the gas, Q_1 is the heat flow between the gas and the liquid, Q_2 is the heat flow between the gas and the tank wall and V_{gas} is the volume of the gas.

Applying Newton Law, we obtain the heat transfer equation:

$$Q = hA\Delta T \tag{11}$$

Here, h is the coefficient of heat transfer, A is the heat transfer area and ΔT is the difference in temperature.

Since free convection is dominant in the tank interior, the average heat transfer is modeled based on the empirical Nusselt number correlations of the type $Nu = CX^n$ (Rohsenow, 1985).

$$h = \frac{k \bullet Nu}{L} \tag{12}$$

The characteristic length L in the above tanks different values for different heat transfer area.

The Modelica language for dynamic simulations is ideal for modeling PFS. Based on the equations described in this section, we could easily build all the corresponding Modelica model. Combining all the related components in Figure 1, we can develop a liquid rocket PFS as follows:

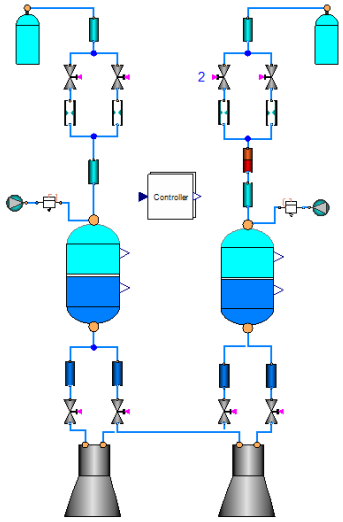


Figure 2 A System Model of PFS in MWorks Modeling View

4 Model-Based Fault Simulation

In the previous sections we have briefly discussed the nominal and faulty behaviors of the basic components. In this section, we describe how to associate the fault modes with the system model and inject the fault mode into a certain model correctly without changing the system topology.

4.1 Fault Mode Library

There are many kinds of fault modes in PFS. Traditionally, fault modes are stored in the literature as

depicted in Table 1. As is shown in Table 1, a single fault mode must be a faulty symptom of a certain product, which belongs to a subsystem. Many items are used to describe the fault mode and each kind of fault mode is tied with a fault parameter, as discussed in Section 3. However, all these items are written in the literature which could not be recognized by the computer. It is necessary to build a standard architect to transfer the literal fault mode into a quantitative fault mode file.

To get a unified description of all kinds of fault modes, we choose a XML file. XML is a hierarchy extensible language featured by self-describing, convenient data processing and easy understanding. It is suitable for the representation of the fault modes of a liquid rocket system. Moreover, it makes it convenient for us to generalize this method to other tools (not limited to MWorks) in the future. According to the requirements of the configuration information in Table 1, the XML file should include at least three contents: the product name, the fault mode description and the fault parameter. Another key element is the trigger time, since most of the fault modes are not triggered at the beginning. The logical hierarchy of the XML file is designed in Code 1.

Table 1 Literal Description of Some Fault Modes

Subsystem	Product	Fault Mode	Fault Parameter
PFS	GasBottle	Leakage	*leak*
	Board	Leakage	*K*
		Clogging	*leak*
	Valve	Always Open	*leak*
		Always Closed	*leak*
	GasPipe	Leakage	*leak*
	LiquidPipe	Leakage	*leak*

Code 1 A Sample of a Certain Fault Mode

```
</Product>
       <Product Type=”gaspipe” Name
="gaspipe" >
       <FaultMode Name="Leakage">
              <Parameter Name="leak"
Value="0" Condition="time>10" />
……
</Product>
```

Code 1 describes the leakage fault mode of a gas pipe according to Table 1. There is a default value for the fault parameter and trigger time, which can be modified before simulation. Moreover, the XML file can be extended to add some new items.

To better modify the XML file from GUI, we have also designed a XML editor on MFC (Petzold, 1998). The XML file can be read and modified by utilizing the DOM (document object module) technology, which is a cross-platform and language-independent convention for representation and interacting objects in HTML, XHTML and XML documents.

Similarly, we could build other fault modes in the XML file just as in Code 1. By extending MWorks (Zhou, 2006), we develop an interface to read the XML file into the simulation environment. The structure of the FML (fault mode library) is depicted in Figure 3.

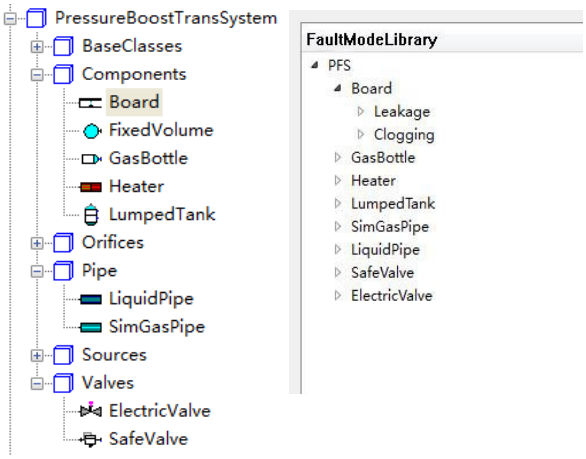


Figure 3 Hierarchical Structure of PFS Library and FML

4.2 Mapping Between Fault Mode and Modelica Model

From the above discussion, we know that faulty behaviors are stored in the Modelica file and fault modes are stored in the XML file. They are implicitly connected via the fault parameter. We need to construct a mechanism to make sure that the fault mode can be injected into the right object.

As the Modelica semantic describes, annotation is an attribute or property containing information associated with some element of a Modelica model and it will not affect the simulation of the model (Peter, 2010). According to the annotation syntax, we introduce a new annotation mapping for associating Modelica model with the fault mode in FML. Take the SimGasPipe leakage fault as an example:

Code 2 SimGasPipe Modelica Code

```
model SimGasPipe
annotation(__MWorks(FaultInfo(ModelName="ga
spipe")));
  extends
DynamicSystem.PressureBoostTransSystem.
BaseClasses.TwoPortsSysGas;
  Real leak = 0.1 "leakage coefficient"
    annotation
(__MWorks(FaultInfo(FaultParameter)));
  parameter Real G(unit="kg/s/pa") = 1000;
equation
  m_flow = G * dp;
  dm_flow = m_flow * leak;
```

As we observe from the code above, the first kind of annotation defines the name of the model which must be consistent with the product type defined in the XML

DOI 10.3384/ecp15118557

file. The defined word *ModelName* is mapped with the string *Type* defined in XML file. The second kind of annotation shows that this variable is a fault parameter and can be changed by fault injection. The defined word *FaultParameter* matches with *Parameter* in the XML file. Both fault modes start with keyword *_MWorks* to make sure that it can be identified by MWorks.

4.3 Fault Injection

Fault injection is a key process in fault simulation. All faults could be injected into a system by changing the system input in some way (Genler, 1991). In the state-space framework, the process is described in the following way:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + Fp(t) \\ y(t) &= Cx(t) + Du(t) + q(t) \end{aligned} \tag{13}$$

Here $u(t)$ and $y(t)$ are the command value of the inputs and the measured value of the outputs, respectively, $p(t)$ and $q(t)$ are the fault vectors and F is the fault entry matrix. Vector p contains the actuator faults, disturbances and input sensor faults. Vector q contains the output sensor faults. The fault entry matrix F is assumed to be known and could be considered as a constant input to the whole system.

Based on the mapping relationship discussed above, we could make fault injection possible by changing the value of the fault parameter in the model. The so-called fault parameter has a meaning different from the definition in Modelica. It can be treated as an input signal to the system as discussed in equation (13). To be able to change the value of the fault parameter during the simulation, we should define it as a variable. Figure 4 helps us better understand the idea.

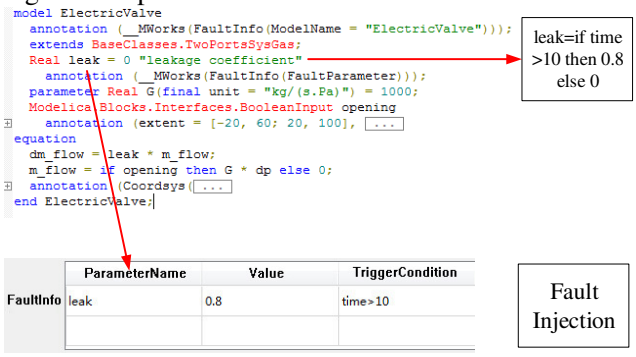


Figure 4 **Fault Injection Operation**

As can be seen from Figure 4, *leak* is the fault parameter defined in the model. The fault mode will be triggered at time 10s. The expression "leak= if time>10 then 0.8 else 0" depicted in Figure 4 is only a pseudo code for easy understanding of the injection mechanism. The platform MWorks will not generate such an equation when compiling the model but just replace the default fault parameter 0 with the new setting parameter 0.8.

4.4 Fault Simulation

To help the user better understand the new fault injection method, we gives a fault simulation procedure as follows:

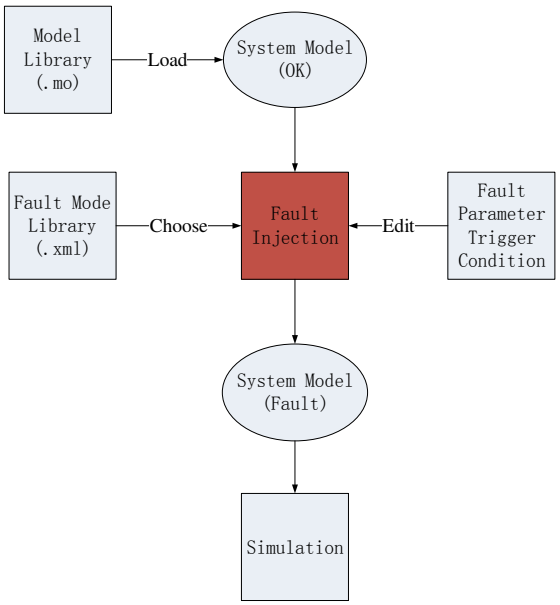


Figure 5 Fault Simulation Process

Firstly, we construct a system model from components in the PFS library. Secondly, we choose the fault mode that is to be simulated from the fault mode library. It is worth to note that there may exist more than one object of the same class in the system for a certain fault mode. For example, there are four SVs, all of which are classed as electric valve in Figure 1. We must make sure that the fault mode is associated with the right object. This problem is resolved by the new annotation mapping. Specifically, MWorks will associate all the related objects in the system model and let the end user choose which object to be injected. Thirdly, we need to specify the fault parameter and fault trigger time. An exception is the nominal situation where there is no need to handle this. Finally, we simulate the system model. At that time, the fault parameter in the model will be replaced by the value we specified if the fault trigger condition is satisfied.

5 Case Study

Based on the system depicted in Figure 2, we first conduct nominal condition simulation to demonstrate the validation of the system model. Then, we show to how to inject several typical fault modes were injected using the method described in Section 5.

5.1 Nominal Condition

Under nominal condition, the propellant pressure should maintain a relatively high level to ensure a safe flight. The simulation result is as follows:

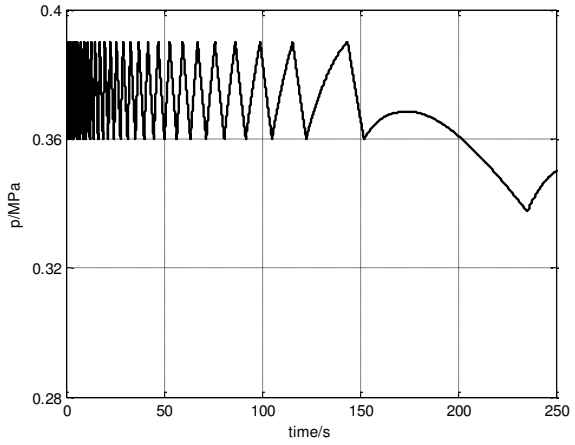


Figure 6 Pressure in the Tank@ Nominal Condition

As we can see from figure 6, the pressure is always within the setting pressure range of SV6 before 150s and SV8 is always closed. After 150s, the gas bottle could not supply enough gas to the propellant tank and the pressure experiences a little drop until 250s. The pressure drops under the minimal control pressure of SV8, causing SV8 opens to increase the pressure of the tank. It can be seen from Figure 6 that the pressure in the tank always keeps a relatively high level and satisfy the requirements of the flight. The system model is well suited for the simulation of PFS.

5.2 Fault Simulation-SV Always Closed

To overcome accidental faults in the flight of a rocket, redundant strategy is always adopted for an easily broken part or a critical part. As is done in Figure 1, two parallel SVs could ensure to some extent the safety of the rocket.

As seen from Figure 7, a fault mode is injected into SV6 at time 50s, which makes the valve always closed. The pressure in the tank drops sharply below the minimum control value of SV8 and it opens to ensure that the pressure keeps oscillating within its setting range. The simulation result shows that the fault injection method is successful and the redundant strategy is helpful in maintaining the tank pressure.

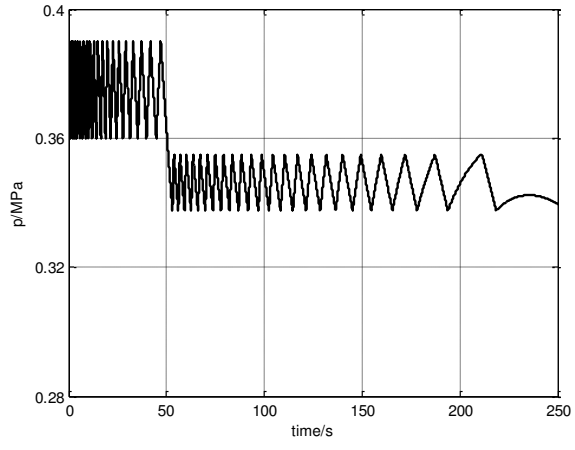


Figure 7 Tank Pressure@ SV6 Fault

5.3 Fault Simulation-Gas Pipe Leakage

Figure 8 shows the result of the tank pressure with a modified leak coefficient. The tank pressure could sustain if the leakage is 20 percent, whereas the pressure drops sharply if the leakage is more than 40 percent. The simulation result demonstrates that the fault injection method is successful and the system has a high level stability even under a fault mode (20% leak-age).

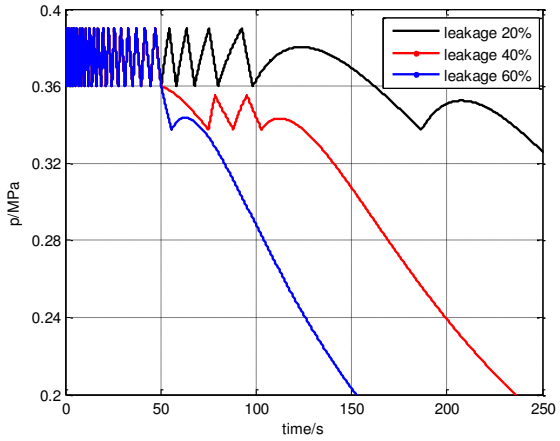


Figure 8 Tank Pressure@ Gas Pipe Leakage

6 Conclusions and Future Work

In this paper, we develop a physics-based model of pressurization and feed systems and analyze the validity of the model. Moreover, we introduce a new annotation mapping mechanism to associate the Modelica model with the fault mode, which lays a foundation for fault injection without modifying the structure of the system model. Standardization of different kinds of fault modes using a XML file is realized as well.

By adjusting the fault parameter of the gas pipe, we obtain simulation results that are consistent with the qualitative analysis, thereby justifying the validity of the new fault injection method. The new method is easy to understand and can support multiple fault modes injection is also supported as well.

Based on the above analysis, we conclude that fault simulation could help us better understand the system and explore the weakness of the system design in advance. The new fault injection method is simpler and more convenient than the traditional methods.

The idea of fault injection is quite general. It is not limited to PFS and can be applied to other domains by simply modifying the fault modes and system model. PFS is just one application and we plan to explore this method further in the future. For example, fault transmission and fault diagnose are on our agenda. We could simulate the process of fault transmission and do some fault diagnose analyses by batch simulation based on current method. Another future research is to combine current work with FMEA efficiently.

DOI 10.3384/ecp15118557

Acknowledgements

The paper is supported by The National High Technology Research and Development Program of China ("863"Program) (No. 2013AA041301)

References

Partola I S. Design of liquid-propellant rocket engines. Journal of Machinery Manufacture and Reliability, 41(6):492-498, 2012.

Daigle M, Foygel M, Smelyanskiy V. Model-based diagnostics for propellant loading systems. Aerospace Conference, 2011 IEEE. pp. 1-11, 2011.

Gao Ming, Niaoqing HU, and Guojun QIN. Object-oriented Modeling and Fault Simulation of Propellant Filling System. Machine Tool & Hydraulics, 37(09):223-226, 2009.

Wang Min, Hu Niaoqing, Qin Guojun. Fault Modeling and Simulation Analysis for LRE Test-bed Filling System. System Simulation, 22(11): 2672-2675, 2010.

Fan Zhongze, Huang Minchao. Fault Simulation of Space Power System in the Operation Process. Journal of National University of Defense Technology, 30(02): 11-15, 2008.

F.L.J. van der Linden. General fault triggering architecture to trigger model faults in Modelica using a standardized blockset. Proceedings of the 10th International Modelica Conference, 2014.

Esposito A. Fluid power with applications. Prentice-Hall International, 2000.

Rohsenow W M, Hartnett J P, Ganic E N. Handbook of heat transfer fundamentals. 1985.

Petzold C. Programming windows. Pearson Education, 1998.

Fan-Li Zhou, Li-Ping Chen, Yi-Zhong Wu, Jian-Wan Ding, Jian-Jun Zhao, Yun-Qing Zhang. MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica. Proceedings of the 5th International Modelica Conference, Vol. 2: 725-731, 2006.

Peter Fritzson. Principles of object-oriented modeling and simulation with Modelica 2.1. John Wiley & Sons. 2010.

Genler J. Analytical Redundancy Methods in Fault Detection and Isolation. Preprints of IFAC/IMACS Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS'91. 1991.