The final authenticated version is available online at https://doi.org/10.1007/978-3-319-64119-5_11.

A model-based extension to HiP-HOPS for dynamic fault propagation studies

Sohag Kabir[1], Yiannis Papadopoulos[1], Martin Walker[1], David Parker[1], Jose Ignacio Aizpurua[2], Jörg Lampe[3] and Erich Rüde[3]

[1]University of Hull, UK, [2]University of Strathclyde, UK, [3]DNV GL SE, Germany

{s.kabir, y.i.papadopoulos, martin.walker, d.j.parker}@hull.ac.uk,
jose.aizpurua@strath.ac.uk, {Joerg.Lampe, Erich.Ruede}@dnvgl.com

Abstract. HiP-HOPS is a model-based approach for assessing the dependability of safety-critical systems. The method combines models, logic, probabilities and nature-inspired algorithms to provide advanced capabilities for design optimisation, requirement allocation and safety argument generation. To deal with dynamic systems, HiP-HOPS has introduced temporal operators and a temporal logic to represent and assess event sequences in component failure modelling. Although this approach has been shown to work, it is not entirely consistent with the way designers tend to express operational dynamics in models which show mode and state sequences. To align HiP-HOPS better with typical design techniques, in this paper, we extend the method with the ability to explicitly consider different modes of operation. With this added capability HiP-HOPS can create and analyse temporal fault trees from architectural models of a system which are augmented with mode information.

Keywords: Model-based safety analysis; Fault Tree Analysis; HiP-HOPS; dynamic systems; temporal fault trees.

1 Introduction

To overcome the limitations of classical approaches to dependability analysis like Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) [1], in the last two decades, research has focused on simplifying dependability analysis by looking at how dependability artefacts can be automatically synthesized from system models. This has led to the field of model-based safety analysis (MBSA) [2]. MBSA approaches offer significant advantages over classical approaches as they utilise software automation and integration with design models to simplify the analysis of complex safety-critical systems. Over the years, several approaches, e.g., Failure Propagation and Transformation Notation (FPTN) [3], Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [4], AltaRica [5], FSAP-NuSMV [6], and AADL with its error annex [7], have been developed to facilitate MBSA of complex systems. An overview of these approaches is available in [8, 9]. These approaches usually combine different classical safety analysis approaches to allow the analysts to perform safety analyses automatically or semi-automatically. For example, HiP-HOPS, a state-of-the-art MBSA approach, enhances an architectural model of a system with logical failure annotations to allow safety studies such as FTA and FMEA. In this way it shows how the failure of a single component or combinations of failures of multiple components can lead to system failures.

Early versions of the HiP-HOPS method used the classical combinatorial model of traditional FTA. In this model, systems failures are caused by logical combinations of component failures as these combine and propagate through the system architecture. However, in modern large-scale and complex systems, system behaviour is dynamic over time. This could be due to their capability to operate in multiple modes, e.g., an aircraft can operate in take-off, flight, and landing modes or it could be because the system behaviour changes in response to different events. This dynamic system behaviour leads to a variety of dynamic failure characteristics such as functionally dependent events and priorities of failure events. It is not only combinations of events that matter but sequences too. As systems are getting more complex and their behaviour becomes more dynamic, capturing this dynamic behaviour and the many possible interactions between the components is necessary for accurate failure modelling.

There are different possibilities to model the dynamic behaviour of a system. On the one hand, it is possible to directly specify the dynamic failure behaviour through dynamic dependability formalisms [9, 10]. One example is Pandora TFTs [10], where dynamic behaviours are modelled using temporal gates and temporal laws are used for qualitative analysis. Pandora can be used in the context of HiP-HOPS for assessing event sequencing in dynamic systems. A difficulty with this approach is that the dynamic operation is not explicitly given in a system design model, but has to be introduced later on in the failure modeling where event sequences are described. This can make application of the method counterintuitive to designers who are used to describing dynamics directly in system models using mode and state diagrams. The difficulty can be overcome by modelling dynamic behaviour in state automata linked to an architectural model of the system and by synthesizing fault trees by traversing the combined model [11]. However, in this approach important information related to the sequencing of events is eventually lost, as the resultant fault trees are combinatorial and do not have temporal semantics. This, however, is not ideal since there are circumstances where the order of two or more events is significant and changes the effects of failure and the capability to recover [10].

The main contribution of this paper is the proposal of a dynamic fault propagation approach which extends the HiP-HOPS technique with explicit representation of system modes and states. The approach generates Pandora temporal fault trees which represent accurately the dynamic failure behaviour of the system without any loss of the significance that the sequencing of events may have. The approach has been illustrated on a model of a twin-engine aircraft fuel distribution system.

2 Background

2.1 An overview of the HiP-HOPS Technique

Hierarchically Performed Hazard Origin & Propagation Studies or HiP-HOPS [4] is one of the more advanced and well supported compositional model-based safety analysis techniques. It can automatically generate fault trees and FMEA tables from extended system models, as well as perform quantitative analysis on the fault trees. It also has the ability to perform multi-objective optimisation of the system models [12]. It can semi-automatically allocate safety requirements to the system components in the form of Safety Integrity Levels (SILs) which automates some of the processes for the ASIL allocation specified in ISO 26262.

The approach consists of three main phases: a) system modelling and failure annotation b) fault tree synthesis and c) fault tree analysis and FMEA synthesis

The system modelling and failure annotation phase allows analysts to provide information to the HiP-HOPS tool on how the different system components are interconnected and how they can fail. The architectural model of the system shows the interconnections between the components of the system and the architecture can be arranged hierarchically, i.e., the system consists of different subsystems and subsystems have their own components. Modelling and annotation of the system with dependability information can be done using popular modelling tools like Matlab Simulink or SimulationX. The dependability related information includes component failure modes and Boolean expressions for output deviations, which describe how a component can fail and how it responds to failures that occur in other parts of the system. The expression for the output deviations show how the deviations in the component outputs can be caused either by the internal failure of that component or by corresponding deviations in the component's input. Such deviations can be user defined but typically include omission (O) of output, unexpected commission (C) of output, incorrect output, or too late or early arrival of output [13] (see Fig. 1). If available, quantitative data can also be entered to facilitate quantitative analysis in a later phase through parametric distribution functions (e.g. failure rate or scale and shape parameters of exponential and Weibull distributions, respectively). Note that while annotating components, HiP-HOPS considers that a component has a fixed set of nominal and failure behaviour, and these behaviours do not change over time. For instance, consider the component shown in Fig. 1, where the annotation of the output deviation of component A is shown in Table 1.

Once the components in the system model are annotated with failure expressions, the annotated model is used by HiP-HOPS to synthesize fault trees. This process starts with a deviation of system output (top event) and traverses the system architecture backwards, i.e., from the system level outputs to the component level failures, to examine the propagation of failures through connections between components. In this way the process traverses the whole architecture and combines the local fault trees from the individual components until no connected components remain. The result is a single fault tree (or set of fault trees) which represents all the possible combinations of component failure that can lead to the system failure.

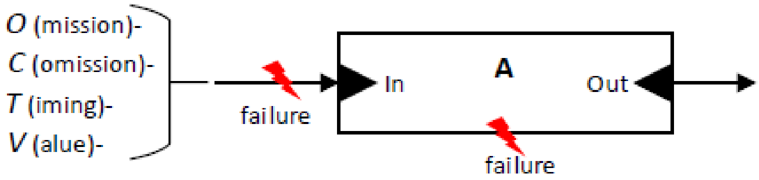

Fig. 1. An example component

Table 1. Annotation of component with static behaviour in HiP-HOPS

Component Name	Output Deviation	Failure Expression
A	O-Out	A.Fail + O-A.In

In the final phase, the synthesised fault trees are analysed both qualitatively and quantitatively. Qualitative analysis results in minimal cut sets (MCSs), which represent the smallest combinations of failure events that can cause the system failure. In addition to that, FMEA tables are generated automatically showing the connections between component failures and system failures. In quantitative analysis, probability of system failure is estimated based on the failure rate/probability of the basic events.

Generally, temporal dependencies among the events are not considered in FTA. However, HiP-HOPS is able to consider them using Pandora temporal fault trees (TFTs) [10]. Pandora uses temporal gates such as Priority-AND (PAND) and Priority-OR (POR) to represent temporal relations among events. The PAND gate represents a sequence between events *X* and *Y* where event *X* must occur before event *Y*, but both the events must occur. The POR gate also represents a sequence between the events, but it specifies an ordered disjunction rather than an ordered conjunction, i.e., event *X* must occur before event *Y* if event *Y* occurs at all. In this paper, the symbols '◁' and '|' are used to represent PAND and POR operation respectively in logical expressions. Additionally, '+' and '.' are used to represent logical OR and AND operations.

2.2 Dynamic Behaviour and Challenges in Dependability Analysis

In modern systems, big tasks are often divided into smaller tasks and are processed in different stages of the operation. In this way, resources are utilised in a sequence of different stages, and in each of those stages, a set of different functions are performed to complete the overall task. For example, the operation of the Aircraft Fuelling Systems (AFS) in modern aircraft can be divided into modes, whereby some of the operations may take place before the flight and some may take place during the flight. Throughout the process, at any particular point in time, some of the system components may act as active components and some others may act as passive components. By active at a point in time, we mean those components which are engaged in system operation at that particular time. On the other hand, inactive components are those which are idle or switched off, i.e., not involved in any operation at that point in time and waiting to be reactivated by the system.

Sometimes a system may have to perform a set of variable functions and, to facilitate this, a variable configuration of the system is obtained by deliberately activating and deactivating a selected number of components. A second scenario could be that a system is performing a fixed set of functions, and in the presence of a failure, the system may sacrifice some of its non-critical functions and go to a degraded operational mode by only doing the critical functions with a limited number of components with a different configuration. Additionally, to make the safety critical system tolerant to faults, many systems have fault tolerance strategies built in. As part of such a fault tolerant strategy, in the presence of faults, systems may reconfigure by using spare (cold or hot) components to respond to the faults and continue the nominal behaviour.

If we want to analyse such a system with techniques like HiP-HOPS, we will soon be faced with difficulties caused by the dynamic behaviour of the system. Temporal fault trees capture temporal dependencies, but for multi-state systems, it is difficult to precisely define the nominal behaviour of the system because it has different behaviours in different modes. Therefore, it is equally difficult to define the potential deviations from the nominal behaviour. Another thing to note is that different selections of components are activated and deactivated to obtain a desirable configuration; therefore some of the components may be irrelevant in some of the modes, and thus so are their failure modes. As a result, it is a challenge to take this mode dependent behaviour into account and represent it in an understandable and manageable format to facilitate dynamic failure propagation studies.

3 Dynamic Fault Propagation Studies using HiP-HOPS

3.1 Representing dynamic behaviour using mode charts

As already mentioned, we consider that in the presence of failure a system can behave dynamically by reconfiguring itself to deliver a variable set of functions or a single set of function with some alternative configurations. That means the configuration of the system may be dependent on the mode in which the system is operating, i.e., a distinct configuration/architecture can be associated with a distinct mode of operation. We propose to use mode charts [14] to represent the functional, dynamic behaviour of the systems, where each mode will represent a distinct configuration and the transition conditions will be the events associated with the component failures. Please note that in future work this concept will be applied in a more general sense, i.e., by also considering events that can transition the system state between operational modes; which is out of the scope of this paper.

A mode chart M could be formally defined as:

$$M = (Q, \Sigma, \delta, q_0) \qquad (1)$$

where Q is the set of all possible modes, Σ is the set of all possible events, δ is the transition function $\delta\colon Q \times \Sigma \to \mathrm{Q}$, and q_0 is the initial mode. The initial mode represents the fully functional architecture of the system where all the system components are operative and all functionality of the system is provided. Each of the other modes represents a degraded architecture (a distinct configuration) which is formed due to the presence of some failure, however, this architecture is still able to provide system functionality. We make the distinction between these modes based on the criticality of the configurations they represent:

- *Critical mode*: any further component failure will result in the system failure.
- *Non-critical mode*: the system failure cannot not be reached directly and further configurations can be formed from the present configuration.

Fig. 2 shows different modes, M_i, connected via transition events, T_i, which cause mode changes. According to the criticality of the modes, they are classified as non-critical modes = {M_1, M_2, M_3} and critical modes = {M_4, M_5, M_6}.

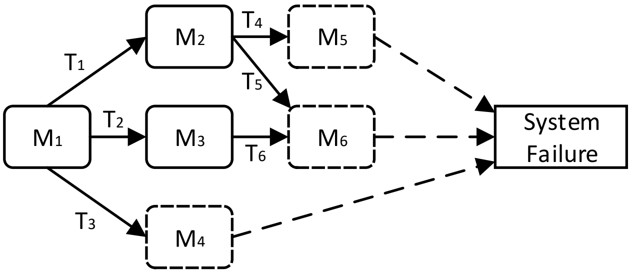

Fig. 2. Concept of mode based behaviour modelling; dashed and solid rounded rectangles represent the *critical* and *non-critical* modes respectively

To describe the dynamic behaviour of a system using mode charts, we would have to identify: (i) all possible functional modes, (ii) all possible conditions that can trigger transitions between those functional modes, and (iii) the *critical* and the *non-critical* modes.

3.2 Annotation of mode based behaviour of components

As mentioned earlier, for static fault propagation studies, the HiP-HOPS technique considers the system architecture as static and annotates the systems components with a fixed set of failure behaviours. These behaviours are considered the *default* behaviours of the components. However, for dynamic fault propagation studies, we need to annotate the components with mode-based behaviour. In this paper, the system components are regarded as non-repairable and have defined failure behaviour for different *critical modes* as the system failure could be reached directly from those modes. The components are non-repairable to ensure that the mode chart will be loop free, i.e., a directed acyclic graph.

Note that a component does not have to have a failure behaviour for all the critical modes because it may be (i) inactive in a particular mode, (ii) failed prior to entering a mode, or (iii) masked due to the failure of other components. In the first case the failure behaviour of that component is irrelevant in this mode. In the second case, the component has already failed before coming to the present mode; therefore, the failure behaviour of the component is already addressed in any of the prior modes. In the third case, the component itself is not failed; however, its activity does not have any effect in the system because of some other reason, e.g., failure of other components.

Table 2. Example of mode-based failure annotation of component

Component	Output	Failure Expression		
Name	**Deviation**	**Mode M_1**	**Mode M_2**	**Mode M_3**
A	O-Out	$E_1 + E_2 + E_3$	N/A	$E_4. E_5$

For example, let us consider that in a functional mode chart there are three critical modes M_1, M_2, and M_3 respectively; the component A is active in mode M_1 and M_3, but not in mode M_2. Therefore, we have to define the failure behaviour of component A only for modes M_1 and M_3. The annotation can be represented in tabular format as displayed in Table 2, where E_i denotes the i-th failure event of component A. If the failure specification of a component is the same in all its modes, the mode-based behavior reduces to their *default* behaviour as defined in the static analysis.

3.3 Synthesis and Analysis of Annotated System Models

Once the mode chart of the system behaviour and the mode based failure data have been defined, the mode chart and the annotated architectures can then be synthesised using the HiP-HOPS technique. This phase operates by examining how the failure of components propagates through system architecture and through different modes in the mode chart to cause system failure. Therefore, the first task of this phase is to identify the parts of the system model that act as the system outputs, and then define system failures (top events of the fault trees) for each of the *critical* modes in the mode chart. The top event of a mode specific fault tree is represented in the following form:

`Output_Deviation_Name`<*mode_name*>

where the *mode_name* inside the angle brackets defines the mode from which the causes for output deviation defined by the `Output_Deviation_Name` are required to be derived. Similarly, mode specific basic events can be named as: `event_name`<*mode_name*>.

The system operation modes denote non-overlapping system states. Accordingly, the system failure condition is defined as the disjunction of the causes of output deviation in all the critical modes. That is:

$$\texttt{O\_D\_X}= \texttt{O\_D\_X}<mode_1> + \texttt{O\_D\_X}<mode_2> + \ldots + \texttt{O\_D\_X}<mode_n> \qquad (2)$$

where `O_D_X` denotes the output deviation X.

In the synthesis process, each top event is considered separately and fault trees are generated using the HiP-HOPS technique by traversing both the mode chart and the system architecture. This process differs from HiP-HOPS' static fault propagation studies in that now sets of fault trees are generated for all the *critical* modes whereas in the static studies with a single operation mode only a single set of fault trees was created to represent the failure behaviour of the whole system. The fault tree synthesis process is now divided into two connected phases:

- *Architecture traversal*: represents the causes of system failure from that particular critical mode (as it is done in static studies).
- *Mode chart traversal*: represents the causes of reaching a particular critical mode from the initial mode.

A graphical overview of the fault tree synthesis process is shown in Fig. 3 where *IM* denotes initial mode, *NM* denotes non-critical modes, and *CM* denotes critical modes. We can see that all the *CM* lead directly to the system failure occurrence. Note that the arrows showing the direction of mode-chart traversal are in opposite direction of mode transitions.

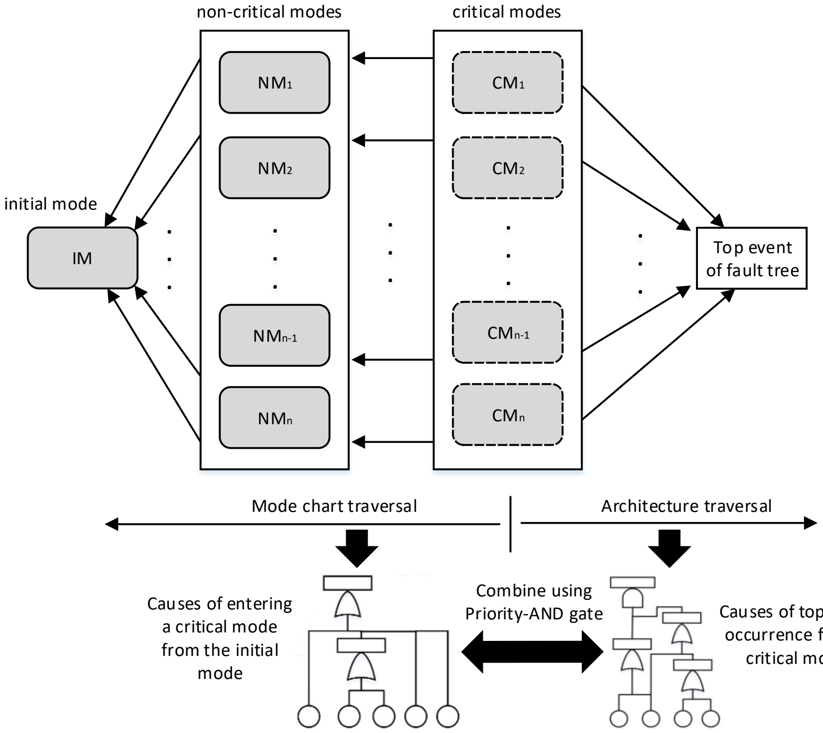

Fig. 3. A graphical overview of the fault tree synthesis process

The two traversal steps generate two interconnected fault trees. The architecture traversal starts with the failure logic of the system output that is defined for this critical mode. It then traverses the static architecture to examine the propagation of failure through the components to the system output. After that, the mode based local fault trees of all involved components are created and this process continues until no connected components remain. Once all the mode based local fault trees are created, they are combined together to obtain a single set of fault trees.

The second set of fault trees is generated by traversing the mode chart. We can consider this as a single source single destination graph traversal problem, where the source is the critical mode under consideration and the destination is the initial mode. This mode-traversal process will be backward from an internal mode towards the initial mode. In every iteration, the process replaces the current mode by its immediately preceding mode(s) and transition conditions from the preceding mode(s) to the current mode. This process will continue until the initial mode is reached. As a result of this process, we obtain all the possible combinations of events (component failures) that cause the system to go to the mode in question from the initial mode. If the initial mode is the critical mode then there is no need to traverse the mode chart.

In order to obtain the complete failure behaviour we need to combine these two sets of fault trees. The system can only fail if it reaches the critical mode of operation first and then from the critical mode to the system failure. Hence, when combining fault tree models generated from mode and architecture traversals, we need to maintain the sequence between them. We can use a PAND gate to combine these two sets of fault trees and define the system failure caused by the mode *i* denoted top-event, $\mathrm{TE_i}$:

$$TE_i = FTA_i \lhd FTA_{architecture} \qquad (3)$$

where FTA_i denotes the fault tree obtained from the mode-chart traversal for the mode *i* and $FTA_{architecture}$ denotes the fault tree obtained from the architectural traversal.

When fault trees for all the critical modes are obtained, they can be linked with the OR logic to obtain the complete failure behaviour of the system. Let us assume that the system has *N* critical modes, then the system failure, *TE*, is defined as:

$$TE=TE_1+\ldots+TE_i+\ldots+TE_N \qquad (4)$$

where TE_i is defined in Eq. (3).

Qualitative analysis could be performed on the Eq. (4) so as to remove redundant events and minimise the expression into a set of minimal cut sequences (MCSQs). MCSQs are the smallest sequence of events that are necessary and sufficient to cause the top event. In this paper, Pandora temporal fault trees are used to illustrate the idea and the methodologies proposed by Walker [10] to obtain MCSQs are applied. After minimization, the quantitative analysis of MCSQs can be performed using the approaches described in [15, 16], however, it is out of scope of this paper.

4 Case study

To illustrate the idea of dynamic fault propagation studies, we use the case study of a hypothetical twin engine aircraft fuel distribution system, shown in Fig. 4.

The system has two fuel tanks TL (Tank Left) and TR (Tank Right); three valves VL (Valve Left), VR (Valve Right), and VC (Valve Centre); two pumps PL (Pump

Left) and PR (Pump Right). Under normal operating conditions, pump PL provides fuel to the Left Engine from tank TL through valve VL and pump PR provides fuel to the Right Engine from tank TR through valve VR. We can denote this as **M_TLTR** mode of system operation and in this mode, the valve VC is kept closed. Now, if we hypothesise a failure such that VR is stuck closed, then fuel flow to the Right Engine from TR is stopped. In this condition, the system can reconfigure itself by opening valve VC, hence continue fuel flow to the Right Engine from TL. We denote this as **M_TL** mode. Similarly, the system can operate in **M_TR** mode by providing fuel to Left Engine from TR in the condition that VL is stuck closed.

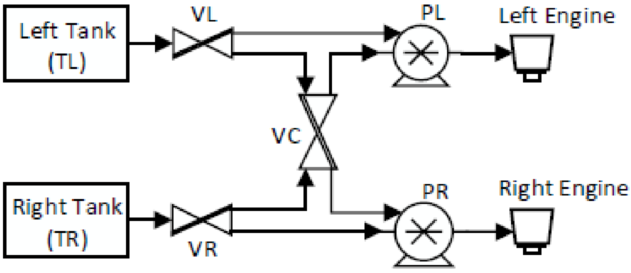

Fig. 4. Architecture of twin engine aircraft fuel system

In the first stage of dynamic fault propagation studies, we need to annotate the components in the system architecture with mode based behaviour. After that, we have to identify the system output. Provision of fuel to each engine could be considered as the system output, and thus failure to provide fuel to any of the engines could be considered as a hazardous condition. Thus, failures of the engines are not considered here. As the fuel to the Left Engine and the Right Engine is provided in a similar fashion with the opposite set of components, for brevity, we concentrate on the failure of the system to provide fuel to the Left Engine alone.

As mentioned earlier, the system can operate in M_TLTR, M_TL, and M_TR modes. From the architecture in Fig. 4, we can see that the failure of pump PL will cause no fuel flow to the left engine in any mode, hence failure of pump PL can be considered as a single point of failure. For this reason, all the modes are considered as *critical* modes as described in Section 3.1. The mode-based annotations of the system components are shown in Table 3. In this table, the value N/A means that the behaviour of the component is not applicable (relevant) in this mode because it has no activity in this mode. We also need to define the mode chart. M_TLTR is the initial mode where all the system components are available. M_TL and M_TR are two degraded modes where the system can provide functionality, in this case, provision of fuel to the left engine, with reduced number of components. A transition from M_TLTR mode to M_TL mode will happen when fuel flow through VR will stop (i.e., O-VR.Out) and the system will enter to M_TR mode from M_TLTR mode when fuel flow through VL stops (i.e., O-VL.Out) (see Fig. 5).

In Table 3, ‘O-’ stands for *Omission*. It can also be seen that failure expressions of some components are the same in different modes. For example, failure expressions for VL is `VL.Fail + O-TL.Out` for both M_TLTR and M_TL modes. However, we treat them differently under the assumption that the probability of failure of a component can be different in different modes due to the change in workloads in different modes, e.g. see [17]. Consider, for instance, each of the engines of the system in Fig. 4 consumes *X* litres of fuel per hour. Therefore, while the system operates in M_TLTR mode *X* litres of fuel flow through both VL and VR. Now, if for some reason VR gets stuck closed then the system will switch to M_TL mode, meaning that the left tank will provide fuel to both engines. This results in double fuel flow from left tank through the VL, which means the workload on valve VL and tank TL get doubled, and this in turn affects the failure probability of these components. As a result, although having the same failure expression in different modes, the behaviour is still treated differently due to the change in failure probability.

Table 3. Mode-based annotations of the components of system in Fig. 4

Component	**Output Deviations**	**Failure Expression**		
		M_TLTR	**M_TL**	**M_TR**
Left Engine	O-Out	O-PL.Out	O-PL.Out	O-PL.Out
PL	O-Out	PL.Fail	PL.Fail + O-VL.Out	PL.Fail + O-VC.Out
PR	O-Out	PR.Fail	PR.Fail + O-VC.Out	PR.Fail + O-VR.Out
VL	O-Out	VL.Fail+ O-TL.Out	VL.Fail + O-TL.Out	N/A
VR	O-Out	VR.Fail+ O-TR.Out	N/A	VR.Fail + O-TR.Out
VC	O-Out	N/A	VC.Fail + O-VL.Out	VC.Fail + O-VR.Out
TL	O-Out	TL.Empty+ TL.Block	TL.Empty + TL.Block	N/A
TR	O-Out	TR.Empty+ TR.Block	N/A	TR.Empty + TR.Block

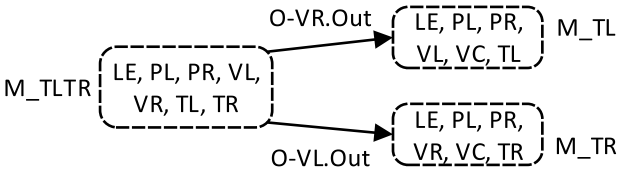

Fig. 5. Mode chart with active components listed in the modes

For fault tree synthesis from the annotated system architecture and mode chart, we defined the output deviation of the system as a disjunction of output deviations in all critical modes as follow:

```
  O-Left_Engine.Out = O-Left_Engine.Out<M_TLTR>
                    + O-Left_Engine.Out<M_TL>
                    + O-Left_Engine.Out<M_TR>
```

In the above expression, O-Left_Engine.Out represents the output deviation of the system, i.e., omission of fuel flow to the left engine. O-Left_Engine.Out<*M_TLTR*>, O-Left_Engine.Out<*M_TL*>, and O-Left_Engine.Out<*M_TR*> represent the output deviation in modes M_TLTR, M_TL, and M_TR respectively, which are essentially the top events of fault trees for the respective modes. Fault trees for each of the modes can be synthesised following the process described in Section 3.3.

Firstly, for the initial mode, M_TLTR, as mentioned in Section 3.3, we need to traverse the system architecture only based on the failure annotations (see Table 3).

```
O-Left_Engine.Out<M_TLTR> = O-PL.Out<M_TLTR>
                          = PL.Fail<M_TLTR>
```

For modes M_TL and M_TR, as they are *critical* but not initial modes, we need to obtain two fault trees. Consider the M_TL mode: to obtain the cause of system failure, we can start with the following expression (cf. Eq. (3)).

```
O-Left_Engine.Out<M_TL>={M_TL} ⊲ O-Left_Engine.Out<M_TL>
```

On the right hand side of the above expression, '⊲' represents a logical PAND operation. The left operand {M_TL} of the PAND operator represents the causes of entering the mode M_TL from the initial mode. On the other hand, the right operand (O-Left_Engine.Out<*M_TL*>) represents the causes of system failure from mode M_TL. Each of these operands represents a top event for two different fault trees and the fault tree can be obtained using the mode chart and architecture traversal process as follows. Data from Table 3 is used in the traversal process.

Mode Chart Traversal:

```
{M_TL}= {M_TLTR}. O-VR.Out<M_TLTR>
= O-VR.Out<M_TLTR>
= VR.Fail<M_TLTR> + O-TR.Out<M_TLTR>
= VR.Fail<M_TLTR> + TR.Empty<M_TLTR> + TR.Block<M_TLTR>
```

Architecture traversal:

```
O-Left_Engine.Out<M_TL> = O-PL.Out<M_TL>
= PL.Fail<M_TL> + O-VL.Out<M_TL>
= PL.Fail<M_TL> + VL.Fail<M_TL> + O-TL.Out<M_TL>
= PL.Fail<M_TL> + VL.Fail<M_TL> + TL.Empty<M_TL>
 + TL.Block<M_TL>
```

Combining the results obtained from above two steps the failure behaviour of the system outputs from the M_TL mode is written as:
O-Left_Engine.Out<*M_TL*>=(VR.Fail<*M_TLTR*>+
TR.Empty<*M_TLTR*> + TR.Block<*M_TLTR*>) ⊲ (PL.Fail<*M_TL*>
+ VL.Fail<*M_TL*> + TL.Empty<*M_TL*> + TL.Block<*M_TL*>)

Similarly, the causes of system failure from mode M_TR can be obtained as:
O-Left_Engine.Out<*M_TR*>=(VL.Fail<*M_TLTR*>+
TL.Empty<*M_TLTR*> + TL.Block<*M_TLTR*>) ⊲ (PL.Fail<*M_TR*>
+ VC.Fail<*M_TR*> +VR.Fail<*M_TR*> + TR.Empty<*M_TR*>
+ TR.Block<*M_TR*>)

Now, the complete failure behaviour of the system can be obtained by taking logical OR of the individual failure behaviour in different modes (cf. Eq. (4)).
O-Left_Engine.Out= PL.Fail<*M_TLTR*>+ [(VR.Fail<*M_TLTR*>
+ TR.Empty<*M_TLTR*> + TR.Block<*M_TLTR*>) ⊲ (PL.Fail<*M_TL*>
+ [VL.Fail<*M_TL*> + TL.Empty<*M_TL*> + TL.Block<*M_TL*>)]
+(VL.Fail<*M_TLTR*>+ TL.Empty<*M_TLTR*> + TL.Block<*M_TLTR*>)
⊲ (PL.Fail<*M_TR*> + VC.Fail<*M_TR*> +VR.Fail<*M_TR*>
+ TR.Empty<*M_TR*> + TR.Block<*M_TR*>)]

From a closer look at the architecture of Fig. 4, we can see that the work pattern or workload on pump PL and PR remains the same in all the modes. For this reason, failure behaviour of these components can be considered to be mode independent, i.e., PL.Fail<*M_TLTR*> ⇔ PL.Fail<*M_TL*> ⇔ PL.Fail<*M_TR*> ⇔ PL.Fail. Therefore, the above expression can be written as:
O-Left_Engine.Out= PL.Fail+ [(VR.Fail<*M_TLTR*>
+ TR.Empty<*M_TLTR*> + TR.Block<*M_TLTR*>) ⊲ (PL.Fail
+ VL.Fail<*M_TL*> + TL.Empty<*M_TL*> + TL.Block<*M_TL*>)]
+ [(VL.Fail<*M_TLTR*>+ TL.Empty<*M_TLTR*> + TL.Block<*M_TLTR*>)
⊲ (PL.Fail + VC.Fail<*M_TR*> +VR.Fail<*M_TR*> + TR.Empty<*M_TR*>
+ TR.Block<*M_TR*>)]

This fault tree expression now shows the causes of omission of fuel to the left engine form all the relevant modes. Using a prototype version of the HiP-HOPS tool, the minimal cut sequences to cause the system failure are calculated, and shown in Table 4. In this table, basic events are replaced by their IDs as: X_1 = PL.Fail, X_2 = VL.Fail<*M_TLTR*>, X_3 = VL.Fail<*M_TL*>, X_4 = VR.Fail<*M_TLTR*>, X_5 = VR.Fail<*M_TR*>, X_6 = VC.Fail<*M_TR*>, X_7 = TL.Empty<*M_TLTR*>, X_8 = TL.Block<*M_TLTR*>, X_9 = TL.Empty<*M_TL*>, X_{10} = TL.Block<*M_TL*>, X_{11} = TR.Empty<*M_TLTR*>, X_{12} = TR.Block<*M_TLTR*>, X_{13} = TR.Empty<*M_TR*>, X_{14}= TR.Block<*M_TR*>.

Table 4. Minimal cut sequences that can cause the system failure in Fig. 4.

MCSQs	MCSQs
X_1	$X_2 \mid X_6 . X_2 \lhd X_5 . X_2 \mid X_{13} . X_2 \mid X_{14}$
$X_4 \lhd X_3 . X_4 \mid X_9 . X_4 \mid X_{10}$	$X_2 \mid X_6 . X_2 \mid X_5 . X_2 \lhd X_{13} . X_2 \mid X_{14}$
$X_4 \mid X_3 . X_4 \lhd X_9 . X_4 \mid X_{10}$	$X_2 \mid X_6 . X_2 \mid X_5 . X_2 \mid X_{13} . X_2 \lhd X_{14}$
$X_4 \mid X_3 . X_4 \mid X_9 . X_4 \lhd X_{10}$	$X_7 \lhd X_6 . X_7 \mid X_5 . X_7 \mid X_{13} . X_7 \mid X_{14}$
$X_{11} \lhd X_3 . X_{11} \mid X_9 . X_{11} \mid X_{10}$	$X_7 \mid X_6 . X_7 \lhd X_5 . X_7 \mid X_{13} . X_7 \mid X_{14}$
$X_{11} \mid X_3 . X_{11} \lhd X_9 . X_{11} \mid X_{10}$	$X_7 \mid X_6 . X_7 \mid X_5 . X_7 \lhd X_{13} . X_7 \mid X_{14}$
$X_{11} \mid X_3 . X_{11} \mid X_9 . X_{11} \lhd X_{10}$	$X_7 \mid X_6 . X_7 \mid X_5 . X_7 \mid X_{13} . X_7 \lhd X_{14}$
$X_{12} \lhd X_3 . X_{12} \mid X_9 . X_{12} \mid X_{10}$	$X_8 \lhd X_6 . X_8 \mid X_5 . X_8 \mid X_{13} . X_8 \mid X_{14}$
$X_{12} \mid X_3 . X_{12} \lhd X_9 . X_{12} \mid X_{10}$	$X_8 \mid X_6 . X_8 \lhd X_5 . X_8 \mid X_{13} . X_8 \mid X_{14}$
$X_{12} \mid X_3 . X_{12} \mid X_9 . X_{12} \lhd X_{10}$	$X_8 \mid X_6 . X_8 \mid X_5 . X_8 \lhd X_{13} . X_8 \mid X_{14}$
$X_2 \lhd X_6 . X_2 \mid X_5 . X_2 \mid X_{13} . X_2 \mid X_{14}$	$X_8 \mid X_6 . X_8 \mid X_5 . X_8 \mid X_{13} . X_8 \lhd X_{14}$

5 Conclusion

In this paper, we pointed out that the dynamic behaviour of systems makes it difficult to precisely define the nominal and failure behaviour of systems, thus complicating dependability analysis processes. As a potential remedy to the above problem, we propose the use of mode charts to define dynamic behaviour of systems, and subsequently annotate the components in the system architecture with their mode based behaviour. The proposed approach extends the state-of-the-art model-based dependability analysis approach, HiP-HOPS, by extending the existing phases of the approach for dynamic fault propagation studies.

The annotation phase has been extended by annotating system components with mode-based dynamic behaviour. The synthesis phase has been extended by providing ways to generate temporal fault trees by examining the system model and how the failure of components propagates through the system architecture and the different modes in the mode chart to cause system failure. Finally, in the analysis phase, minimal cut sequences are generated by analysing the temporal fault trees. As a whole, this extension to HiP-HOPS combines the advantages of the existing HiP-HOPS approach — semi-automatic generation of system-wide failure propagation information from an annotated system model — with the benefits of forms of representation better suited to dynamic systems, such as mode charts. This combination allows designers to model more complex dynamic scenarios in a more intuitive way than simply using temporal expressions and logic. It also allows them to perform compositional dynamic dependability analysis of complex systems by generating temporal fault trees. This work enriches the semantics of HiP-HOPS and has the potential to be combined with the other advanced features of HiP-HOPS, such as architecture optimisation, maintenance, safety requirement allocation, and safety case generation for dynamic systems. However, the scalability of this approach for analysis of large-scale systems is yet to be verified. Some of our current work is focused on continuing development of the techniques as part of HiP-HOPS tool.

Acknowledgments

This work was partly funded by the DEIS H2020 project (Grant Agreement 732242).

References

1. Vesely, W.E., Stamatelatos, M., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault tree handbook with aerospace applications, NASA office of safety and mission assurance, Washington DC, (2002).
2. Joshi, A., Heimdahl, M.P.E., Miller, S.P., Whalen, M.W.: Model-based Safety Analysis. NASA Technical Report , Hampton, VA, USA (2006).
3. Fenelon, P., McDermid, J.A.: An Integrated Toolset For Software Safety Analysis. J. Systems and Software. 21, 279–290 (1993).
4. Papadopoulos, Y., Mcdermid, J.A.: Hierarchically Performed Hazard Origin and Propagation Studies. SAFECOMP'99, pp. 139–152 (1999).
5. Arnold, A., Point, G., Griffault, A., Rauzy, A.: The AltaRica formalism for describing concurrent systems. Fundam. Informaticae. 40, 109–124 (2000).
6. Bozzano, M., Villafiorita, A.: The FSAP/NuSMV-SA Safety Analysis Platform. Int. J. Softw. Tools Technol. Transf. - Spec. Sect. Adv. Autom. Verif. Crit. Syst. 9, 5–24 (2007).
7. Feiler, P., Rugina, A.: Dependability Modeling with the Architecture Analysis & Design Language (AADL). Technical Report, Carnegie Mellon University (2007).
8. Aizpurua, J.I., Muxika, E.: Model-Based Design of Dependable Systems : Limitations and Evolution of Analysis and Verification Approaches. Int. J. Adv. Secur. 6, 12–31 (2013).
9. Sharvia, S., Kabir, S., Walker, M., Papadopoulos, Y.: Model-based dependability analysis: State-of-the-art, challenges, and future outlook. In: Software Quality Assurance: In Large Scale and Complex Software-intensive Systems. pp. 251–278 (2015).
10. Walker, M.: Pandora: A Logic for the Qualitative Analysis of Temporal Fault Trees, PhD Thesis, University of Hull (2009).
11. Rauzy, A.: Mode automata and their compilation into fault trees. Reliab. Eng. Syst. Saf. 78, 1–12 (2002).
12. Papadopoulos, Y., Walker, M., Parker, D., Sharvia, S., Bottaci, L., Kabir, S., Azevedo, L., Sorokos, I.: A synthesis of logic and bio-inspired techniques in the design of dependable systems. Annual Reviews in Control. 41, 170–182 (2016).
13. Papadopoulos, Y., Mcdermid, J., Sasse, R., Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure, RESS 71, 229–247 (2001).
14. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Failure diagnosis using discrete-event models. IEEE Trans. Control Syst. Technol. 4, 105–124 (1996).
15. Kabir, S., Walker, M., Papadopoulos, Y.: Quantitative evaluation of Pandora Temporal Fault Trees via Petri Nets. IFAC-PapersOnLine. 48, 458–463 (2015).
16. Kabir, S., Walker, M., Papadopoulos, Y.: Reliability Analysis of Dynamic Systems by Translating Temporal Fault Trees into Bayesian Networks. Model-Based Safety and Assessment. pp. 96–109. Springer International Publishing, Cham (2014).
17. Labeau, P.E., Smidts, C., Swaminathan, S.: Dynamic reliability: towards an integrated platform for probabilistic risk assessment. Reliab. Eng. Syst. Saf. 68, 219–254 (2000).