Augmenting a Hazard Analysis Method with Error Propagation Information for Safety-Critical Systems

by Fryad M. Rashid and John D. McGregor
Clemson, South Carolina

Safety-critical system development requires an explicit design to manage component failures and unanticipated conditions of abnormal interaction between system components as hazards that affect the safety and reliability of the system. The potential effects of residual hazards in the operational system context must be reduced to an acceptable level of risk. System reliability focuses on providing continued operational capability in spite of failures. System safety focuses on unsafe conditions because of failures and unpredicted interactions between system components.

Researchers in reliability and safety have developed techniques for the component level and system level. For example, methods have been proposed to perform system-level hazard analysis, such as Fault Tree Analysis (FTA) [Ref. 3], Hazards and Operability Analysis (HAZOP) [Ref. 4], and Failure Modes and Effect Analysis (FMEA) [Ref. 5]. In addition, these compositional approaches are proposed based on the component level, such as Failure Propagation Transformation Notation (FPTN) [Ref. 6], Hierarchically Performed Hazard Origin and Propagation Studies (HipHOPS) [Ref. 7], Component Fault Trees (CFT) [Ref. 8], and Fault Propagation and Transformation Calculus (FPTC) [Ref. 9].

These techniques have been applied in the real world to safety-critical systems for many years. According to our experience in applying hazard analysis methods on different types of systems, these hazard analysis techniques cannot describe the dynamic error behavior of the system, system states and the transitions of the system, and error propagations among system components. Because of this, traditional hazard analysis techniques depend on decomposition of the system with respect to the hierarchy of failure effects, rather than the system's architectural model. The overall goal of this research is to develop procedures that augment existing hazard analysis techniques with error propagation information and state machines to support modeling and analyzing dynamic error behavior.

The increasing complexity and criticality of modern computing systems are driving the need for enhanced hazard analysis methods. A major objective of our research is to create a deeper safety analysis in the engineering practice by augmenting the recent hazard analysis technique, System-Theoretic Process Analysis (STPA), with the specification of error propagation and dynamic support contributors. This augmentation is used to support modeling dynamic error behavior of the system during hazard analysis, and to identify potential internal failures of the major components during application of STPA.

The significant contribution of this work is a method to augment STPA with error propagation information and dynamic support contributors. This will allow us to identify additional hazards based on different criteria for existing application examples, identify additional unsafe control actions in different analyses, identify general and specific causes for the unsafe control actions, and blend safety constraints with the system's architecture to build a safe product. The proposed method is used to help stakeholders or safety analysts during hazard analysis to consider error behavior of a component, which may lead to inadequate control actions, and to verify each path in the form of three-way interactions among components in the feedback control loop to analyze the trace of each hazard that may lead to an accident. We illustrate the method by describing several scenarios and model Scenario 1 by using the Architecture Analysis and Design Language (AADL) supported by the Open Source Architectural Tool Environment (OSATE) to develop a safety architecture representation.

Background

In this section, we give background on recent hazard analysis approaches, error ontologies that provide information about error types, and architecture fault modeling that helps us understand error propagation.

Systems-Theoretic Process Analysis (STPA)

STPA is a top-down hazard analysis approach built on the Systems-Theoretic Accident Model and Process (STAMP). The major idea behind this approach is to investigate an accident before it occurs. The main goal is to identify potential causes of accidents — that is, scenarios that may lead to losses — so they can be controlled or eliminated in the system design or operations before damage happens. Simply put, it provides scenarios to control and mitigate the hazards in the system design. The method consists of four steps to provide scenarios [Refs. 10-12]:

1. The stakeholder establishes fundamental analyses to identify accidents and the hazards associated with those accidents.
2. The stakeholder designs a feedback control loop for the system to identify major components such as sensors, controllers, actuators and the controlled process.
3. The stakeholder identifies unsafe control actions that could lead to hazardous states. The stakeholder can use the control table to identify unsafe control actions and can translate it to corresponding safety constraints.
4. The stakeholder identifies causal factors for the unsafe control actions. The safety analyst determines how each hazardous control action could occur by identifying the process model variables for the controller in the feedback control loop and analyzing each path to find out how each hazard could occur.

Error Ontology

The error ontology, defined in the Error Annex for AADL, represents error types in a hierarchical structure to support hazard analysis. It provides the concept of error type to characterize the types of errors to be propagated. The ontology presents an error event for an activated fault type and presents an error behavior state for each failure mode type. The error type can be described as a categorical label to characterize the type of error declarations in error propagations, error events, error behavior states, error flows and error containment. Also, the label is used to characterize condition declarations for state transitions, detections and outgoing error propagations. Stakeholders can use error types to describe how the components could fail and to associate with error events. For instance, the effect of a sensor failure might be that it dispatches an incorrect reading (value error), it misses a reading (item omission) or it does not provide any readings (service omission). These effects can be caused by various factors, such as overheating, radiation and low power. The error ontology classifies errors into six major error types, including service errors, value errors, timing errors, replication errors, concurrency errors and access control errors [Refs. 1 and 13]. We give a brief description of error ontology in Table 1.

Architecture Fault Modeling

Architecture fault modeling and analysis supports automated safety, reliability and security analyses from the same architecture model to ensure consistency across analysis results. The AADL error model annex supports a bottom-up safety analysis method to support

Table 1 — Error Ontology of Major Error Types.

Error Type	Description
Service Errors	Represent errors that are related to delivering service for items. Service errors differ from omission errors, which represent no service delivered for the items, and commission errors, which represent unexpected service provided for the items.
Value Errors	Represent errors that are related to the value domain of a service. Value errors differ from value errors for individual service items such as incorrect value, value error for sequence of service items such as bounded value change, and value errors related to the service as a whole like out of calibration.
Timing Errors	Represent errors that are related to the time domain of a service. Timing errors differ from timing errors for individual service items like early/late item delivery, timing errors for sequence of service items like rate errors, and timing errors related to the service as a whole like early/delayed services.
Replication Errors	Represent errors that are related to delivery of replicated services. For example, replicated service items delivered for one recipient or to multiple recipients.
Concurrency Errors	Represent errors that are related to the behavior of concurrent systems, such as executing tasks concurrently to access shared resources. Here, errors are distinguished between race condition errors, and mutual exclusion errors.
Access Control Errors	Represent errors that are related to the operation of access control services, such as authentication and authorization errors.

architecture fault modeling by enabling annotation of an architecture model with fault occurrence, resulting failure, and fault propagation behavior to address dependability concerns in safety-critical systems. This approach can support architecture fault modeling at three levels of abstraction [Refs. 1 and 15]:

- **Fault propagation across the system:** Fault propagation and its impact on the system, its operational environment and among components within a system. This level supports safety analysis in the form of hazard identification and fault impact analysis.
- **Fault and recovery behavior of components:** Fault identification and its occurrence, its manifestation in the component as a failure mode, the effect of incoming propagation on failure mode, the propagation of failure mode and incoming propagation as outgoing propagation, and the ability of the component to detect and recover itself. This level supports safety analysis in probabilistic reliability and availability analysis.
- **Compositional abstraction of fault models:** Related to the fault models of the system components to abstract the fault model of the system. This level supports safety analysis in scalable compositional fault analysis.

Method

We extend each step in the STPA technique with error propagation information and dynamic error behavior. In addition, we add a final step to merge safety aspects of the system into the system's architecture. The steps are:

1. **Identify hazards using different criteria.** This step involves identifying accidents based on the system operational context. The error ontology provides guidance in identifying hazards.
2. **Build control structures with contributors.** This step includes the construction of a feedback control loop with finite state machines for describing dynamic behavior of the system and adds error propagation specifications across the system to analyze the trace of the hazards that may lead to accidents.
3. **Identify unsafe control actions using tracing.** This step helps to identify unsafe control actions based on error propagations tracing. It identifies the error behavior of a component that can lead to inadequate control actions that could become an unsafe action. This step also helps to identify any error flow for which a corresponding safety constraint needs to be created to mitigate identified hazards.
4. **Identify specific causes.** This step helps to identify causes for the unsafe control actions in the feedback control loop system. Generally, it needs to select the component first and then specifically look for the causes which relate internally and externally to the component.
5. **Develop safety architecture.** Safety architecture is implemented based on the safety constraints identified in the previous step. It blends safety aspects of the system into the overall system architecture.

We created this augmentation to help the stakeholder or the safety analyst identify and evaluate dysfunctional behavior of the system during hazard analysis. The main goal of analyzing the behavior of the system is to identify hazardous control actions by considering the specification of error propagations across the system, and operating states of the system, which can have an effect on control actions. Using the characteristics of event-driven models and error propagation information in the safety analysis can assist in providing an effective way to annotate and assess all possible paths in the feedback control loop system to identify hazards. In this method, steps 1 through 4 can identify the source of errors as hazardous situations and their impact on the other components during the operational environment of the system. Step 5 helps to enhance the safety of the system by feeding the identified hazardous situations or propagated error events into the system's architecture to absorb unsafe actions. Here, we describe the augmented process by using several scenarios and examples to support our method.

The heart of the STPA approach is a feedback control loop to analyze the safety of the system. We want to improve the method to support deeper safety analysis for system development. Figure 1 shows the feedback control loop with error propagation and finite-state machine models. Figure 1 consists of four component types: sensors, controllers, actuators and the controlled process. These components have incoming and outgoing ports used to send and receive data or information about different types of error events. The connections among components represent the nominal control flow, as well as the error propagation path. The error propagation path follows port connections from sensors to controllers, controllers to actuators, actuators to the controlled process, and controlled process to the sensors. First, the sensors measure the values of attributes and send them to the controller. Each sensor has two states, operational and failed. Second, the controller acquires information about the state of the process from measured variables and controlled variables, and uses this information for initial action by manipulating controlled variables to maintain the operational process within predefined limits. The controller is used to regulate the process variables and send commands to

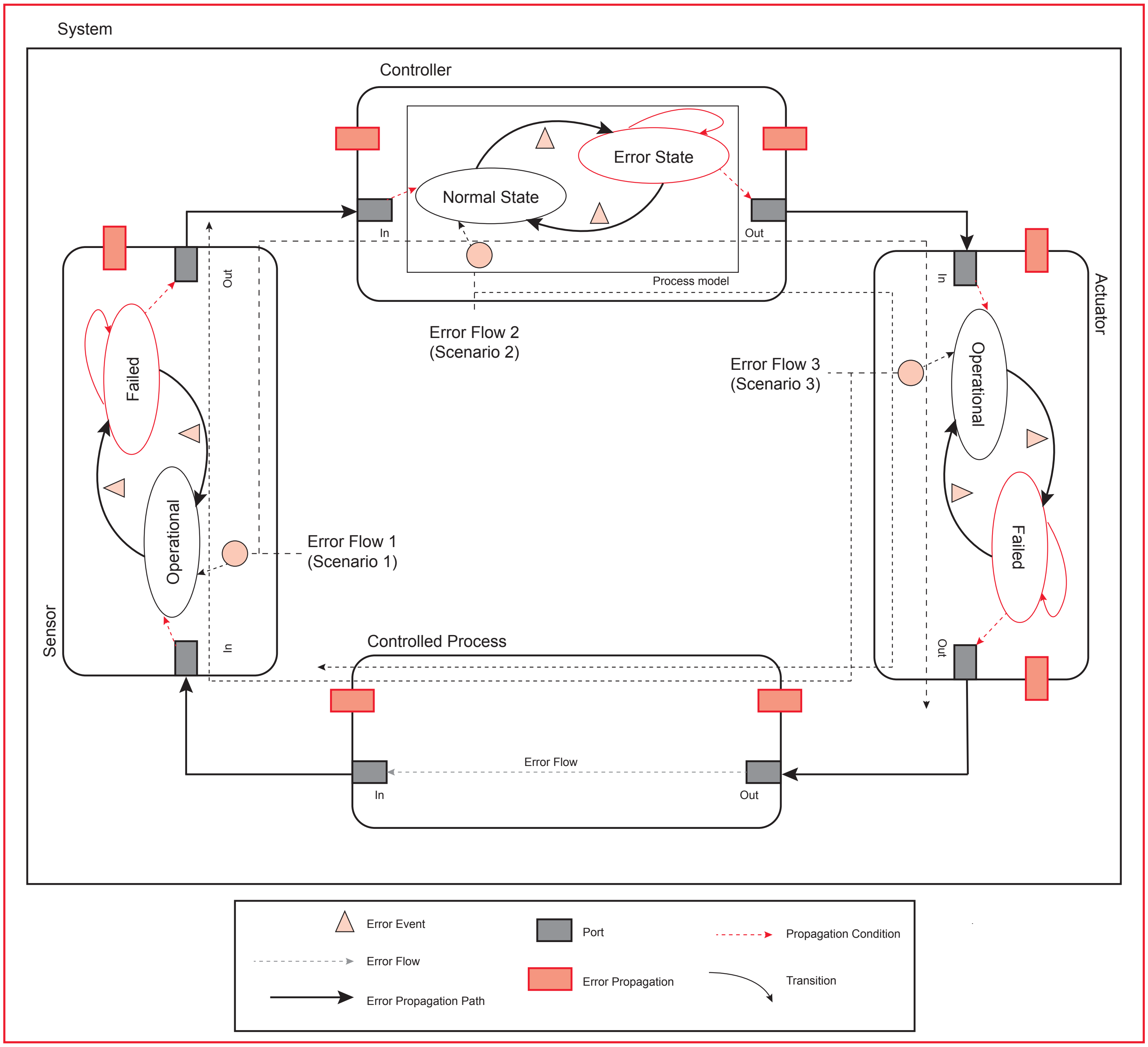


Figure 1 — Feedback Control Loop Augmented with Error Propagation and Finite State Machines.

the actuator. The controller consists of a process model, which is used to present variables and their values. The controller has two states: normal and error. The normal state shows the status of the variables in normal operation. The error state shows abnormal values of the variables. Third, the actuator follows the controller instructions to execute commands. The actuator has two states, operational and failed. Fourth, the controlled process is used to show processes inside the controller. The error flow passes errors inside the component from an incoming port to an outgoing port [Refs. 1 and 2]. In Figure 1, we show three scenarios to support our method.

Scenario 1

Scenario 1 begins with the sensor in the operational state. The failed state is entered when the sensor detects an internal failure. This means that the sensor becomes a source of error propagation. In this scenario, we show how error propagates from the sensor to the actuator, and becomes hazardous to the system. In the event that the sensor detects an internal failure, the sensor will be a source of error events. The error continuously affects the normal operational state of the sensor. If the sensor is able to recover, the error will not propagate to the next component. But, if the sensor does not recover, the error propagates through the outgoing port and along the propagation path to the controller. The controller receives the error through its incoming port and puts it in the process model to process. In this case, the process model does not understand the propagated error because it holds the value of the difference between the observed value and the specified value. For that reason, the propagated error becomes an error event, which may lead to changing from the normal state to the error state. If the controller is able to handle the error event and would not be hazardous to the system, the controller will automatically go back to its normal state. But, if the controller is not able to solve it, the situation could become hazardous for the system, such as sending an inadequate or ineffective command to the actuator. The actuator acts based on orders from the controller. Therefore, the actuator will execute an inadequate action. This kind of action can be considered an unsafe control action because it is made based on propagated error. If we want to diagnose the cause of the unsafe action, we must go back to the system controller because we expect that the controller did not control the process. However, in this situation the controller does not have any faults. In this case, we must go back a step further to find the specific or exact cause of the fault. We must then analyze the component interactions in the feedback control loop system in a three-way interaction, rather than a two-way interaction. Finally, we show that the propagated error from sensor to actuator has three important effects in the system. First, it affects the state of the sensor itself. Second, it affects the decision of the controller. Third, it affects the actuator to perform ineffective action. This illustration allows us to identify the source of the hazard by back-tracing for the error.

We provide an example to support this scenario and show how our method helps to identify additional informative features to predict hazardous situations for an existing application example in References 2 and 13. For example, what will happen if the sensor in adaptive cruise control (ACC) estimates the incorrect values for speed and distance of the car in front of its car (e.g., the sensor estimates that it is close enough, but in reality, it is not) during driving because of an internal failure? Basically, when the component has an internal failure, the result is propagation. These two values are propagating through the outgoing port of the sensor and along the propagation path to the controller for processing. The controller receives these two values from the incoming port and puts them into the process model for computation. As a result, the controller is performing an incorrect computation because it received incorrect values or abnormal data. The result of the computation is that an incorrect command is sent to the actuator. In this situation, the ACC system warns the driver to apply the brake because the car is too close to the car in front of it. If the driver does not perform that action, the system will do so automatically. Either way, an accident will occur because the car is performing an unacceptable action based on a decision made with incorrect values. In this case, either the driver or the system will apply the brakes in the middle of the road, and the car may be hit by the vehicle behind it. Finally, people may be harmed because of propagating error events from the sensor to the actuator. The result of this example extends the result of References 2 and 13. The following example details the results when using our method:

- Step 1: Error ontology identifies the wrong estimation as incorrect values. The hazard is "incorrect estimation values for the car from the ACC system."
- Step 2: Feedback control loop system has been built from the sensor to the actuator as shown in Figure 1, which is error flow 1 (Scenario 1).
- Step 3: The unsafe control action is that "system applies the brake in the middle of the road." The safety constraint for the unsafe action is that "the system must not apply the brake when it has incorrect values."
- Step 4: The general cause for the unsafe control action is that "the sensor has internal failure" and the specific cause is "propagating incorrect values as an error event from sensor to actuator."
- Step 5: This is shown in the developed safety architecture subsection shown in Figure 2.

Scenario 2

Scenario 2 begins with the controller in the normal state. The error state is entered when the controller detects an internal failure. The generated error does not propagate outward; it stays within the controller. Different types of errors propagate through the outgoing port to the actuator, instead of the generated error. For example, the controller generates an (out of range) error, but it sends a (no data) error to the actuator because the controller is unable to handle containment errors.

In the case of internal failure, the controller becomes a source that generates errors for the next component. The generated error affects the normal operational state continuously. If the controller is able to recover , the error will not propagate. But, if it is not able to recover, the controller changes the the error to another type and then propagates it through an outgoing port and along the propagation path to the actuator. In this hazardous situation, for each propagated error the actuator is executing a different type of unacceptable action because the controller is a complex system that has many variables with many types of errors. For instance, if these errors are generated in the controller (i.e., "stuck value" error, "out of range" error or "out of calibration" error), the actuator is performing an unacceptable action for the transformed error (i.e., no value, bad value, or incorrect value) because the controller changed the type of the unhandled error. After that, the controlled process receives the propagated error from the actuator as an inadequate or ineffective action through the error propagation path, and then the error passes through the error flow inside of the controlled process to the outgoing port. The inadequate or ineffective action of the actuator is not able to control processes within a predefined limit in the controlled process. Therefore, the output of the controlled process would be an incorrect or inaccurate measurement — or would have a feedback delay. For example, if the input of the controlled process has a delayed operation, its output certainly would have a feedback delay. These are events contributed to the system hazard because of error propagations.

Finally, we illustrated that the propagated error from controller to controlled process has three important effects on the system. First, it affects the controller decision. Second, the actuator performs an inadequate action. Third, it affects the controlled process by selecting the inappropriate process for the action. This illustration allows us to identify the source of the hazard by back-tracing for the error.

The following example supports Scenario 2: We have an automated door control system for a train in Reference 2. We need to extend the same example to improve the safety of the system from better to best. For instance, what would happen if the controller in an automated door control system for a train (ADCST) sends "0" value instead of "1" value because of internal failure to select the doors to open, and to allow people to move out when the train is stopped completely at the station platform? Certainly, if the controller has an internal failure, it produces an internal error. The error becomes an event and leads to changing the normal operational state of the process model to the error state. If the controller is able to handle the error event, it is not going to be a hazardous situation. The controller will autonomously go back to its normal state. But if it is not able to handle the error event, the error will be transformed into an outgoing propagation type from controller to actuator, such as the actuator getting a "0" value error instead of a "1." As a result, the actuator will perform the wrong action. For example, it might select the right-side doors processes in the controlled process to execute instead of the left side doors. This can definitely lead to people being harmed because the system guides people in the wrong direction. Our method can be expanded to add the following information for the (ADCST) example in Reference 2.

- Step 1: Error ontology identifies wrong selection as incorrect values. Now, the hazard is "wrong side selection to open the doors at the station platform."
- Step 2: Feedback control loop has been built from the controller to the controlled process as shown in Figure 1, which is error flow 2 (Scenario 2).
- Step 3: The unsafe control action is that "system selects right side doors to open instead of left side: That is an unsafe action." The safety constraint for the unsafe action is that "the system must not open the doors when it has incorrect values."
- Step 4: The general cause for the unsafe control action is that "the controller has an internal failure," and the specific cause is "the propagating transformed error event from controller to controlled process."
- Step 5: Space constraints for this document do not allow us to show the developed safety architecture for this example.

Scenario 3

Scenario 3 begins with the actuator in the operational state. The failed state is entered when the actuator detects an internal failure. The actuator receives a command from the controller through the incoming port, but the command is not executed as the controller intended because of the actuator's internal failure. The result of the internal failure is propagation of an error. The actuator produces an error internally. The generated error affects the received commands because the error event results in changing the operational state to failed state. If the actuator is able to solve this problem, it does not impact the controller's command. If the actuator is not able to handle the problem, this could become a hazardous situation. This situation impacts the actuator's output, such as having a delay in operation — in other words, a timing error. The delayed operation directly affects the controlled process because the timing error passes through the error flow to the output port. The output of the controlled process becomes an input to the sensor, but it has a feedback delay because of propagating the timing error. At the same time, the output of the sensor becomes an input to the controller, but it also has a feedback delay due to propagating the timing error. Therefore, the propagated error from the actuator to the sensor has three important effects on the system: First, it affects the actuator state. Second, it affects the controlled process by having a delayed time to select the appropriate process for the action. Third, it affects the sensor by having delayed time to obtain measured values. This illustration allows us to identify the source of the hazard by doing the back-tracing for the error.

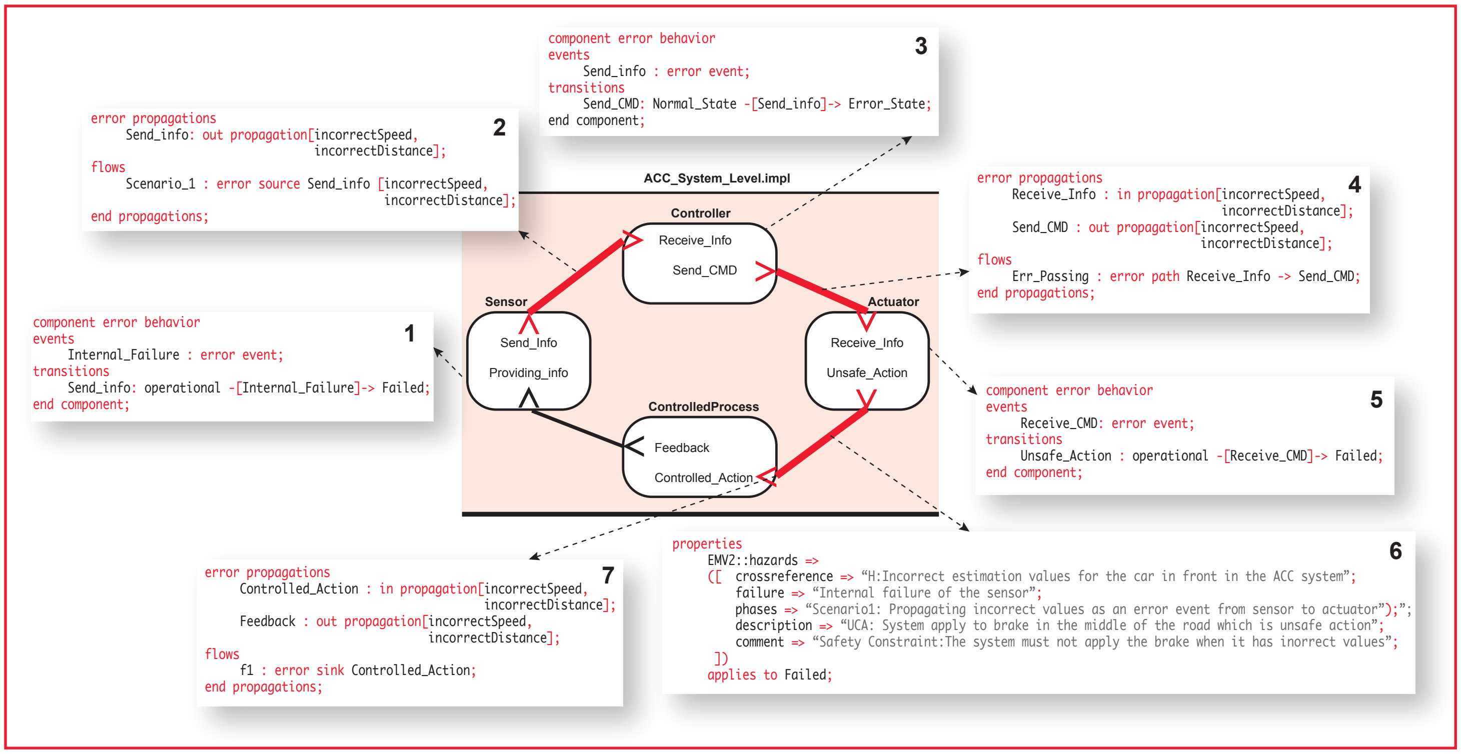
As an example for this scenario, we use the real safety-critical embedded system for a medical device — the pacemaker. The pacemaker is used to regulate an abnormal heartbeat. It has two main essential tasks: sensing and pacing. In pacing, it paces the heart in case the heart's own rhythm is irregular or too slow. In sensing, it monitors the heart's natural electrical activity. If the pacemaker senses a "normal" heartbeat, it will not stimulate the heart. Now, according to our method, we need to specify major components of the pacemaker, such as the controller (DCM: Device-Controller Monitor), the actuator (PG: Pulse Generator), the controlled process (heart) and the sensor (electrode/lead) [Refs.14 and 15]. We can connect the components as shown in Figure 1. Our illustration starts from the actuator to the sensor's output as shown in Figure 1 error flow 3 (Scenario 3: PG → heart → electrode/lead). If the sensor detects that the heart needs pacing, the DCM sends the command to the PG to send a pulse to the heart. The PG receives the command, but it is not executed directly because it has an internal failure. In this case, the produced error inside the PG continuously affects the normal operational state of the PG and directly impacts the DCM's command. If the PG is able to solve the error, no hazardous situation would result. But if it is not able to solve the error the situation will be hazardous for the patient. It means that some errors may occur in the pacemaker, such as a timing error (delayed service or late delivery) for this situation, causing the patient's heart not to get delivered therapy. The timing error leads to a lack of control of the heart rate. This feedback delay of the heart becomes an input to the sensor. The sensor sends this status to the controller. The controller process model makes a comparison between the sensing value and the threshold value to correct this problem. If the controller decides to send another command based on different values to the PG to send one more pulse to the heart, the problem will not be solved because the heart gets another late delivery of therapy. The health of the patient is not going to be better because of accumulating late delivery therapy, which does not help to increase the slow heart rate. Therefore, the internal failure of the PG directly impacts the patient's heart because of timing error propagation from actuator to the sensor. The result of this example is shown as follows:

- **Step 1:** Error ontology identifies timing errors, such as late delivery. Now the hazard is "The pacemaker is not working properly."
- **Step 2:** The feedback control loop has been built from the actuator to the sensor as shown in Figure 1 which is error flow 3 (Scenario 3).
- **Step 3:** The unsafe control action is that "The pacemaker does not provide an electrical pulse when it's required." The safety constraint for the unsafe action is that "The pacemaker should provide an electrical pulse whenever needed."
- **Step 4:** The general cause for the unsafe control action is that "The actuator has an internal failure," and the specific cause is "the propagation timing error from the actuator to the sensor."
- **Step 5:** Space constraints do not allow us to show the developed safety architecture for this example.

Develop Safety Architecture

Step 5 is the development of safety architecture (the proposed method) and is used to feed the previous steps into the system's architecture to absorb the unsafe control actions, identify hazards in the early system design, provide safety requirements for each hazard and provide specific causes for each unsafe action. In fact, developing the safety architecture is directly combined with the hazard analysis process and architecture design efforts. In this section, we examine Scenario 1 for the ACC system's architecture. The Architecture Analysis and Design Language (AADL), supported by Open Source Architectural Tool Environment (OSATE), is used to develop the STPA pattern, and to augment it with error propagation information and dynamic behavior contributors. We record the information, such as error events, propagated errors, states, and transitions in the error model (EMV2) for the major components. The numbers in Figure 2 are equal to the steps shown as follows:

1. The sensor's internal failure is recorded as an error event. It changes the operational state of the sensor to a failed state. This failure affects the sensor's reading values as incorrect values for the car in front (e.g., incorrect speed and incorrect distance).
2. The result of internal failure is the propagation of incorrect values for the speed and the distance throughout the event port and the propagation path to the controller for processing.
3. The controller receives these two values from the incoming port and starts making computations. But the controller does not understand the propagated incorrect values. For that reason, the propagated error becomes an error event and results in changing the normal state of the controller to an error state. The result of the computation is that an incorrect command is sent to the actuator because the controller received incorrect values or abnormal data.
4. The command is sent to the actuator through the control flow, as well as through the propagation path.
5. The actuator receives the command. Then, the system warns the driver to apply the brake because the car is close enough to the car in front of it, which is not actually true. If the driver does not perform that action, the system will automatically do so. Either way, an accident will occur because the car is per-



```
component error behavior
events
     Internal_Failure : error event;
transitions
     Send_info: operational -[Internal_Failure]-> Failed;
end component;
```

```
error propagations
     Send_info: out propagation[incorrectSpeed,
                               incorrectDistance];
flows
     Scenario_1 : error source Send_info [incorrectSpeed,
                                         incorrectDistance];
end propagations;
```

```
component error behavior
events
     Send_info : error event;
transitions
     Send_CMD: Normal_State -[Send_info]-> Error_State;
end component;
```

```
error propagations
     Receive_Info : in propagation[incorrectSpeed,
                                  incorrectDistance];
     Send_CMD : out propagation[incorrectSpeed,
                               incorrectDistance];
flows
     Err_Passing : error path Receive_Info -> Send_CMD;
end propagations;
```

```
component error behavior
events
     Receive_CMD: error event;
transitions
     Unsafe_Action : operational -[Receive_CMD]-> Failed;
end component;
```

```
properties
     EMV2::hazards =>
     ([  crossreference => "H:Incorrect estimation values for the car in front in the ACC system";
         failure => "Internal failure of the sensor";
         phases => "Scenario1: Propagating incorrect values as an error event from sensor to actuator");";
         description => "UCA: System apply to brake in the middle of the road which is unsafe action";
         comment => "Safety Constraint:The system must not apply the brake when it has inorrect values";
      ])
     applies to Failed;
```

```
error propagations
     Controlled_Action : in propagation[incorrectSpeed,
                                       incorrectDistance];
     Feedback : out propagation[incorrectSpeed,
                               incorrectDistance];
flows
     f1 : error sink Controlled_Action;
end propagations;
```

Figure 2 — Developed Safety Architecture for the ACC System Example.

forming an unsafe action based on a decision made with incorrect values. In this case, either the driver or the system will apply the brakes in the middle of the road, and the car may be hit by the vehicle behind it.

6. The result of the ACC system example is feeding into the Error Model Annexes (EMV2) such as hazard, type of failure, unsafe actions and/or safety constraints.
7. The controlled process incoming port is going to be the sink or destination of the error.

Discussion

In this section, we illustrate the new capabilities of this method:

1. The method helps eliminate unsafe situations through the dynamic structure of the system. The operation of the system is useful because it produces errors that can be traced back to causes.
2. The method is able to identify potential internal failures of the major components in the feedback control loop system to reduce the potential effects of residual hazards in the operational system context.
3. The method is able to show the effect of unsafe interactions, based on the three-way communication format among components and back-tracing for the error.
4. The method helps reduce dependence on human experts in organizations and/or companies to identify extra hazardous situation behavior for the system.
5. The method can predict important informative features, such as finding additional hazardous situations for the existing application examples.
6. Through the method, we are able to answer the following research questions for safety-critical systems:
   - RQ1) How can we improve safety of the system by augmenting a hazard analysis method with error propagation information?
   - RQ2) How can we analyze the internal failures of the components and show the effect of the other components during the hazard analysis process?
   - RQ3) How can we identify dysfunctional behaviors of the components during the hazard analysis process?
   - RQ4) How does this hazard analysis method support dynamic error behavior?

Conclusion and Future Work

We have concluded that this method is different from STPA in using error propagation information and finite machines for the feedback control loop to visualize the actual behavior of the system during hazard analysis. Therefore, this method found unsafe control actions based on dynamic error behavior; it found specific hazards based on error ontology; and it found specific causes for the unsafe control action based on three-way interactions and back-tracing for the error when the STPA does not find them for the same example. This does not mean that this method is a replacement for STPA. However, it more effectively analyzes the elements of a safety-critical system. Finally, this method can assist the safety analyst or the stakeholder during STPA to analyze the component interactions in the feedback control loop system in a three-way interaction, rather than a two-way interaction, to identify the effect of the unsafe actions on other components. The future direction of this work is to add an advanced step in the error propagation information into this method. That step would be a composite error behavior state specification of a system with regard to error behavior states of its subsystems or components. For example, taking controller 1 and controller 2 in parallel and putting them into the system to determine how the system deals with propagating errors for one or both of the controllers.

About the Authors

Dr. John D. McGregor is an associate professor emeritus of computer science at Clemson University in Clemson, South Carolina, a visiting scientist at the Software Engineering Institute and a partner in Luminary Software, a software/systems engineering consulting firm. He regularly engages large software development organizations at all levels from strategic to tactical to the concrete. His research interests include highly-reliable software-intensive systems, software product lines, socio-technical ecosystems, model-driven development and software/system architecture. He serves on the program committee of six to 10 conferences per year and researches, writes and practices strategic software engineering. His consulting has included satellite operating systems, telephony infrastructure, cell phones, software certification and software-defined radios. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001).

Fryad M. Rashid is a Ph.D. student in Clemson University's Computer Science Department. He is doing research in the Strategic Software Engineering Research Group (SSERG). He is interested in developing safety analysis methods for safety-critical systems, cyber-physical systems and real-time embedded systems. He has used AADL/OSATE for system development in software/system architecture, and software verification and validation.

References

1. Feiler, Peter, et al. "Architecture Fault Modeling and Analysis with the Error Model Annex, Version 2," Technical Report: CMU/SEI-2016-TR-009, Carnegie Mellon University/Software Engineering Institute, Pittsburgh, Pennsylvania, June 2016.
2. Leveson, Nancy. "An STPA Primer Version 1," MIT Publications, Cambridge, Massachusetts, http://sunnyday.mit.edu/STPA-Primer-v0.pdf, August 2013.
3. Baig Ahmed, et al. "Reliability Analysis Using Fault Tree Analysis: Review," *International Journal of Chemical Engineering and Applications*, Vol. 4, No. 3, June 2013.
4. Nolan, Dennis. *Safety and Security Review for the Process Industries, Fourth Edition*, Elsevier, Amsterdam, The Netherlands 2015.
5. Mikulak, Raymond, et al. *The Basics of FMEA, Second Edition*, Taylor & Francis Group, Abingdon, U.K., 2009.
6. Fenelon, Peter, et al. "An Integrated Toolset for Software Safety Analysis," *The Journal of Systems and Software*, Vol. 21(3), pp. 279–290, June 1993.
7. Papadopoulos, Y. "Analysis and Synthesis of the Behavior of Complex Programmable Electronic Systems in Conditions of Failure," *Reliability Engineering and System Safety*, Vol. 71(3), pp. 229-247, Elsevier, 2001.
8. Kaiser B., et al. "A New Component Concept for Fault Trees," *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software*, Vol. 33, 37-46, ACM, 2003.
9. Wallace Malcolm. "Modular Architectural Representation and Analysis of Fault Propagation and Transformation," *Electronic Notes in Theoretical Computer Science*, 141(3):53–71, Elsevier, 2005.
10. Abdulkhaleq, Asim and Stefan Wagner. "Integrating State Machine Analysis with System-Theoretic Process Analysis," *LNI Proceeding of Software Engineering Workshop Band*, Vol. P-215, pp. 501-514, Köllen Druck+Verlag GmbH, 2013.
11. Hommes Qi. "Applying STPA to Automotive Adaptive Cruise Control System," MIT Publications, Cambridge, Massachusetts, 2012.
12. Leveson, Nancy. "Engineering a Safer World: Systems Thinking Applied to Safety," MIT Press, Cambridge, Massachusetts, 2011.
13. Feiler Peter. "Architecture-Led Safety Analysis of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System," Technical Report: CMU-SEI-2015-SR-032, Carnegie Mellon University/Software Engineering Institute, Pittsburgh, Pennsylvania, December 2015.
14. National Heart, Lung, and Blood Institute. "Explore Pacemakers," https://www.nhlbi.nih.gov/health/health-topics/topics/pace, February 28, 2012.
15. Standard Pacemaker. "Pacemaker System Specification," Software Quality Research Laboratory (SQRL), Boston Scientific, January 3, 2007.