Toward a UCM-based Approach for Recovering System Availability Requirements from Execution Traces

Jameleddine Hassine[1] and Abdelwahab Hamou-Lhadj[2]

[1] Department of Information and Computer Science
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
jhassine@kfupm.edu.sa
[2] Electrical and Computer Engineering Department
Concordia University, Montréal, Canada
abdelw@ece.concordia.ca

Abstract. Software maintenance accounts for a significant proportion of the cost of the software life cycle. Software engineers must spend a considerable amount of time understanding the software system functional attributes and non-functional (e.g., availability, security, etc.) aspects prior to performing a maintenance task. In this paper, we propose a dynamic analysis approach to recover availability requirements from system execution traces. Availability requirements are described and visualized using the Use Case Maps (UCM) language of the ITU-T User Requirements Notation (URN) standard, extended with availability annotations. Our UCM-based approach allows for capturing availability requirement at higher levels of abstraction from low-level execution traces. The resulting availability UCM models can then be analyzed to reveal system availability shortcomings. In order to illustrate and demonstrate the feasibility of the proposed approach, we apply it to a case study of a network implementing the HSRP (Hot Standby Router Protocol) redundancy protocol.

1 Introduction

Software comprehension is an essential part of software maintenance. Gaining a sufficient level of understanding of a software system to perform a maintenance task, is a time consuming and requires studying various software artifacts (e.g., source code, documentation, etc)[1]. However, in practice most of the existing systems have a poor and outdated documentation, if it exists at all. One of the common approaches in getting to understand a software is the study of its runtime behavior, also known as *dynamic analysis* [2]. Dynamic analysis typically comprises the analysis of system behavioral aspects based on data gathered from a running software (e.g., through instrumentation). Dynamic analysis, however, suffers from the size explosion problem of typical execution traces [3]. In fact, executing even a small system may generate a considerably large set of events. Hence, there is a need to find ways to create higher abstractions from low-level traces that can later be mapped to system requirements. To tackle this issue, many abstraction-based techniques have been proposed [4–6], allowing for the grouping of execution points that share certain properties, which results in a more abstract representation of software.

The widespread interest in dynamic analysis techniques provides the major motivation of this research. We, in particular, focus on recovering non-functional requirements, such as availability requirements, from system execution traces. This is particularly important for critical systems to verify that the running implementation supports availability requirements, especially after the system has undergone several ad-hoc maintenance tasks. Avizienis et al. [7] have defined the availability of a system as being the readiness for a correct service. Jalote [8] deemed system availability is built upon the concept of system reliability by adding the notion of recovery, which may be accomplished by fault masking, repair, or component redundancy.

In this paper, we propose the use of to Use Case Maps [9] language, part of the ITU-T User Requirements Notation (URN) standard, as a visual mean to facilitate the capturing of system availability features from execution traces. Previous work [10–14] have considered availability tactics, introduced by Bass et al. [15], as a basis for extending the UCM [9] language with availability annotations. Bass et al. [15] have introduced the notion of tactics as *architectural building blocks* of architectural patterns. These tactics address fault detection, recovery, and prevention.

This paper serves the following purposes:

- It provides an approach based on the high level visual requirements description language Use Case Maps [9] to recover system availability features from execution logs. Using our approach, an analyst can select a particular feature of interest, exercise the system with this feature and analyze the resulting execution trace to determine whether or not availability is taken into account. Although, other visualization techniques can be employed, we have selected the UCM language as our visualization method because it allows for an abstract description of scenarios, that can be allocated to a set of components. Furthermore, through the UCM stub/plugin concept, different levels of abstractions can be considered. The resulting UCM can be later analyzed using the UCM-based availability evaluation technique introduced in [14].
- It extends the set of UCM-based availability features introduced in [12–14] by introducing UCM-based distributed redundancy modeling. The proposed extensions are implemented using metadata within the jUCMNav [16] tool.
- It demonstrates the feasibility of our proposed approach using a case study of a network implementing the Cisco Hot Standby Router Protocol (HSRP) [17].

The remainder of this paper is organized as follows. The next section introduces briefly the availability description features in Use Case Maps. Our proposed approach for the recovery of availability requirements from execution traces is presented in Sect. 3. Section 4 demonstrates the applicability of the proposed approach to the Cisco proprietary Hot Standby Router Protocol (HSRP).

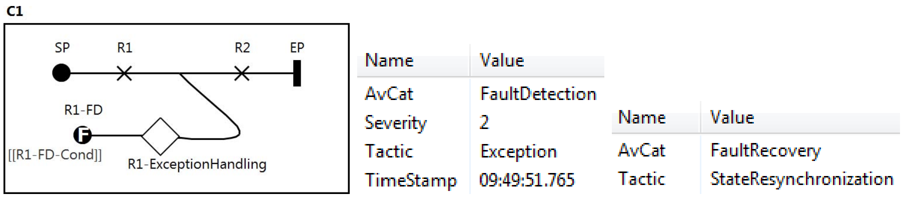
A discussion of the benefits of our approach and a presentation of the threats to validity is provided in Sect. 5. Finally, conclusions and future work are presented in Sect. 6.

2 Describing Availability Requirements in Use Case Maps

In this section, we recall the UCM-based availability requirements descriptions that are relevant to this research. We mainly focus on (1) the implementation of the *exception* tactic, part of the UCM fault detection modeling category, and on (2) the redundancy modeling, part of the UCM fault recovery modeling category. For a detailed description of UCM-based availability features, interested readers are referred to [14], where the UCM-based availability extensions are described using a metamodel.

2.1 Exception Modeling

Exceptions are modeled and handled at the scenario path level. Exceptions may be associated with any responsibility along the UCM scenario execution path. A separate failure scenario path, starting with a failure start point, is used to handle exceptions. The failure path guard condition (e.g., *R1-FD-Cond* in Fig. 1(a)) can be initialized as part of a scenario definition (i.e., scenario triggering condition) or can be modified as part of a the responsibility expression. The handling of the exception, embedded within a static stub (e.g., *R1-ExceptionHandling* in Fig. 1(a)), is generally subject to the implementation of fault recovery tactic through some redundancy means (see Sect. 2.2). Figure 1(c) shows the metadata attributes of a responsibility (within the R1-ExceptionHandling stub) implementing the *StateResynchronization* tactic. After handling *R1* exception, the path continues explicitly with responsibility *R2*.



(a) C1 exception handling scenario

Name	Value
AvCat	FaultDetection
Severity	2
Tactic	Exception
TimeStamp	09:49:51.765

(b) R1 metadata

Name	Value
AvCat	FaultRecovery
Tactic	StateResynchronization

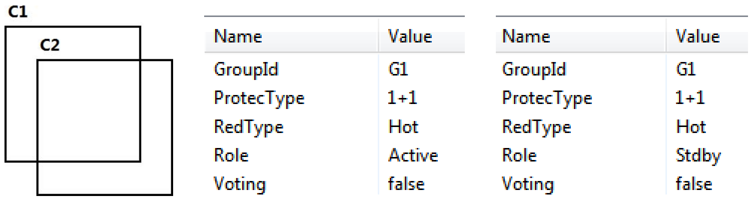
(c) Fault recovery metadata

Fig. 1. UCM exception handling tactic

In addition to the three metadata attributes associated with responsibility *R1* (*AvCat* (specifies the availability category, e.g., *FaultDetection*), *Tactic* (specifies the deployed tactic, e.g., *Exception*), and *Severity* (fault severity, e.g., 1 being the most severe) that have been introduced in previous research [14, 12], we add a *timestamp* attribute to be able to capture the occurrence time of the responsibility (extracted from the log files). Other time-based attributes such as *delay* and *duration*, introduced as part of the Timed Use Case Maps language [18], are not necessary in our context.

2.2 Redundancy Modeling

Fault recovery tactic focuses mainly on redundancy modeling in order to keep the system available in case of the occurrence of a failure. To model redundancy, UCM components are annotated with the following attributes: (1) *GroupID* (identify the group to which a component belongs in a specific redundancy model), (2) *Role* (*active* or *standby* role), (3) *RedundancyType* (specifies the redundancy type, e.g., *hot*, *warm*, or *cold*, (4) *ProtectionType* (denotes the redundancy configuration, e.g., 1+1, 1:N, etc.), and (5) *Voting* (specifies whether a component plays a voting role in a redundancy configuration).



Name	Value
GroupId	G1
ProtecType	1+1
RedType	Hot
Role	Active
Voting	false

Name	Value
GroupId	G1
ProtecType	1+1
RedType	Hot
Role	Stdby
Voting	false

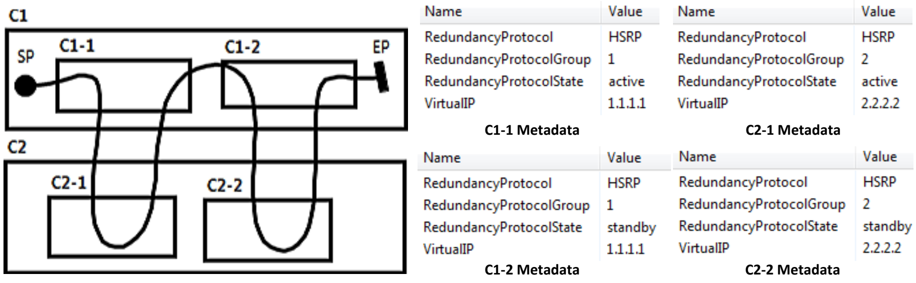
(a) Two redundant components (b) C1 attributes (c) C2 attributes

Fig. 2. UCM node protection

Figure 2 illustrates an example of a system with two components *C1* (*active*) and *C2* (*standby*) participating in a 1+1 hot redundancy configuration. It is worth noting that the above redundancy annotations refer to the initial system configuration state. The operational implications, in case of failure for instance, can be described using the UCM scenario path, e.g., as part of an exception handling path (see Sect. 2.1).

2.3 UCM Distributed Redundancy Modeling

The generic UCM-based annotations describing redundancy [14, 12], presented in the previous section, can be refined to cover redundancy of components that are not physically collocated. Two or more components can be part of a redundancy configuration without being physically on the same device. Such a redundancy can be achieved through a redundancy protocol such as HSRP (Hot Standby Router Protocol) [19] and VRRP (Virtual Router Redundancy Protocol) [20] in IP-based networks.



Name	Value
RedundancyProtocol	HSRP
RedundancyProtocolGroup	1
RedundancyProtocolState	active
VirtualIP	1.1.1.1

C1-1 Metadata

Name	Value
RedundancyProtocol	HSRP
RedundancyProtocolGroup	2
RedundancyProtocolState	active
VirtualIP	2.2.2.2

C2-1 Metadata

Name	Value
RedundancyProtocol	HSRP
RedundancyProtocolGroup	1
RedundancyProtocolState	standby
VirtualIP	1.1.1.1

C1-2 Metadata

Name	Value
RedundancyProtocol	HSRP
RedundancyProtocolGroup	2
RedundancyProtocolState	standby
VirtualIP	2.2.2.2

C2-2 Metadata

(a) Distributed UCM Architecture (b) Component Metadata Attributes

Fig. 3. Distributed UCM architecture implementing more than one redundancy configuration

In order to describe redundancy protocols in UCM, additional metadata attributes need to be incorporated:

- *RedundancyProtocol*: denotes the protocol name, e.g., HSRP, VRRP, etc.
- *RedundancyProtocolGroup*: denotes the redundancy group associated with the redundancy protocol).
- *VirtualIP*: denotes the virtual IP address shared by one or more distributed components.
- *RedundancyProtocolState*: denotes the redundancy protocol state, e.g., active, standby, init, etc.).

Depending on the targeted abstraction level, other relevant metadata attributes may be added like *MacAddress*. Figure 3(a) illustrates a generic UCM architecture with 2 main components C1 and C2. Two HSRP redundancy configurations are described, one for group 1 that involves subcomponents C1-1 (*active* state) and C2-1 (*standby* state), and the other for group 2 that involves subcomponents C1-2 (*standby* state) and C2-2 (*active* state). More details about HSRP can be found in Sect. 4.

3 Recovey of Availability Requirements from Execution Traces

Figure 4 illustrates our proposed approach. The first step consists on collecting system logs (from a single or multiple systems). Typically, a log file is composed of individual log entries ordered in chronological order. Each entry is described as a single line in plain text format and may contain one or more of the following attributes: a time-stamp, the process ID generating the event/error, operation/event prefix, severity of the error, and a brief description of the event/error. Some systems (e.g., Apache and IIS) generate separate log files for *access* and *error*.

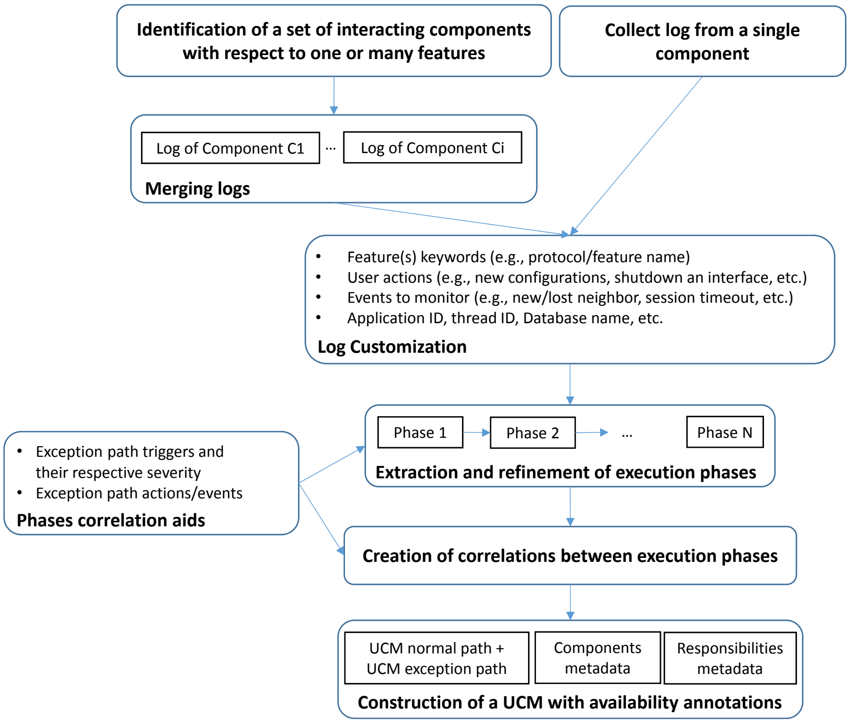

Identification of a set of interacting components with respect to one or many features
Collect log from a single component
Log of Component C1
...
Log of Component Ci
Merging logs
Feature(s) keywords (e.g., protocol/feature name)
User actions (e.g., new configurations, shutdown an interface, etc.)
Events to monitor (e.g., new/lost neighbor, session timeout, etc.)
Application ID, thread ID, Database name, etc.
Log Customization
Phase 1
Phase 2
...
Phase N
Extraction and refinement of execution phases
Exception path triggers and their respective severity
Exception path actions/events
Phases correlation aids
Creation of correlations between execution phases
UCM normal path + UCM exception path
Components metadata
Responsibilities metadata
Construction of a UCM with availability annotations

Fig. 4. Recovery of Availability Requirements from Execution Traces Approach

In case, we are targeting systems with more than one component, a prior knowledge of the possible interactions between the involved components (e.g., protocols used to coordinate the interacting components) is required. After identifying the interacting components, their respective log files are merged and sorted based on timestamps. In order to have a focused analysis of the resulting log, we may reduce its size by applying analyst-defined customization criteria. An analyst may reduce or extend a log window, include or exclude log entries based on features/protocols names, administrator operator actions (e.g., add/remove configuration, shut/unshut network interfaces, etc.), events to monitor (e.g., session timeout, network interfaces state changes, neighbors up/down, etc.). To make an insightful decision, these criteria are applied to the merged log rather than individual logs.

The next step deals with the extraction of the system execution phases. An execution phase is a grouping of a set of log entries (into clusters) based on a predefined set of criteria, such as functionality, component ID, system events, user actions, etc. Our ultimate goal is to be able to map log traces into UCMs (the final step of our approach) using the availability annotations presented in Sect. 2. Given the sequential nature of log file structures, additional analyst input is required in order to distinguish a normal scenario path from an exception path, and to construct correlations between execution phases. Analyst input may include:

- List of potential events/actions/errors/failures triggering the exception path and their respective severity (optional), e.g., shut/unshut a network interface, protocol state changes (down/up), etc. These triggers should be placed in the normal scenario path.
- List of potential events that should be placed in the exception path, e.g., failover, rollback, process restart, HSRP state changes, etc. Typically, an exception path describes the system reaction to an error/failure (i.e., system recovery). Hence, administrator actions should not be placed in the exception path.

Furthermore, an analyst may specify the format and keywords that would help the extraction of components metadata attributes. Section 4 provides an example of component metadata recovery from HSRP log traces. In addition, the following guidelines are developed in order to construct the execution phases and to promote separation of concerns:

- Log entries from different components should be placed into separate phases (i.e., an execution phase cannot span more than one component unless it is part of a component containment configuration).
- Log entries describing different features' events/errors should be placed into separate phases.
- Log entries relative to user actions should be separated from system response log.

It is worth noting that the segmentation of a log into execution phases should not break the causality between different log entries. Finally, the last step consists of mapping the execution phases into UCM models and generating the UCM component related attributes. The following recommendations guide the mapping process:

- Each log entry is mapped to one responsibility.
- An execution phase with more than one responsibility is described using a plugin enclosed within a static stub (named with the name of the execution phase).
- A phase, part of the exception path, having a single responsibility should be enclosed within a static stub.

- Depending on the targeted level of abstraction, sequential stubs bound to the same component and belonging to one path (regular or exception), may be refactored into a static stub.
- Component related information such as the redundancy protocol, the redundancy group, etc., are mapped to component metadata attributes.
- In case two log entries have the same time stamp, their corresponding responsibilities should be enclosed within an AND-Fork and an AND-Join.

4 Case Study: Hot Standby Router Protocol (HSRP)

In what follows, we apply our proposed approach to the HSRP [17] redundancy protocol.

4.1 Hot Standby Router Protocol (HSRP)

Hot Standby Router Protocol (HSRP) is a Cisco proprietary protocol that provides network redundancy for IP networks [17]. By sharing an IP address and a MAC (Layer 2) address, two or more routers can act as a single "virtual" router, known as an HSRP group or a standby group. A single router (i.e., Active router) elected from the group is responsible for forwarding the packets that hosts send to the virtual router. If the Active router fails, the Standby router takes over as the Active router. If the Standby router fails or becomes the Active router, then another router is elected as the Standby router. HSRP has the ability to trigger a fail-over if one or more interfaces on the router go down. For detailed information about HSRP, the reader is referred to RFC 2281 [19].

4.2 Experimental Setup

Figure 5 illustrates our testbed topology, used to implement and collect router logs relative to the HSRP feature. The testbed has been built using the Graphical Network Simulator 3 (GNS3) simulation software [21]. GNS3 allows researchers to emulate complex networks, since it can combine actual devices and virtual devices together. GNS3 supports the Cisco IOS by using Dynamips, a software that emulates Cisco IOS on a PC. In our setup, we have used 4 Cisco c7200 routers (R1, R2, Site1, and Site2) and two Ethernet switches (SW1 and SW2). Two networks are configured (10.10.10.0/24 on the left hand side of the topology, and 10.10.20.0/24 on the right hand side of the topology). Two HSRP groups are configured: Group1 (virtual IP address: 10.10.10.10) on interfaces f0/0 of R1 and R2, and Group2 (virtual IP address: 10.10.20.20) on interfaces f0/1 on R1 and R2. R1 is the active router for Group 1, while R2 is the active router for Group 2.

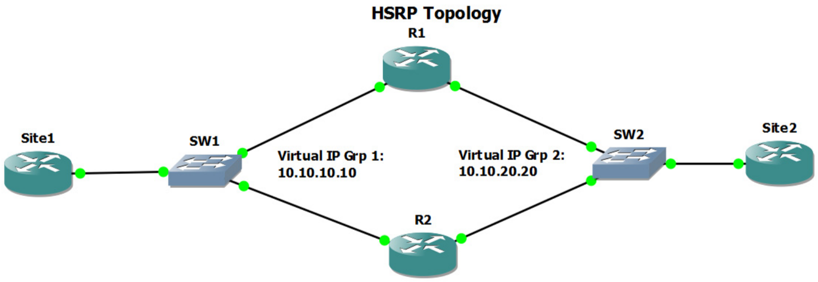


Fig. 5. HSRP Experimental Setup

4.3 Cisco IOS Logging System

Logs can be collected from Cisco IOS routers through console logging (default mode), terminal logging (displays the log messages on VTY lines), buffered logging (use the router's RAM to store logs), syslog server logging (use of external syslog servers for log storage), and SNMP trap logging (send log messages to an external SNMP server).

Any collected log may have one or more components from the following three types:

1. System log messages: They can contain up to 80 characters and a percent sign (%), which follows the optional sequence number or/and time-stamp information, if configured [22]. Messages are displayed in this format:

   seq no:timestamp: %facility-severity-MNEMONIC:description

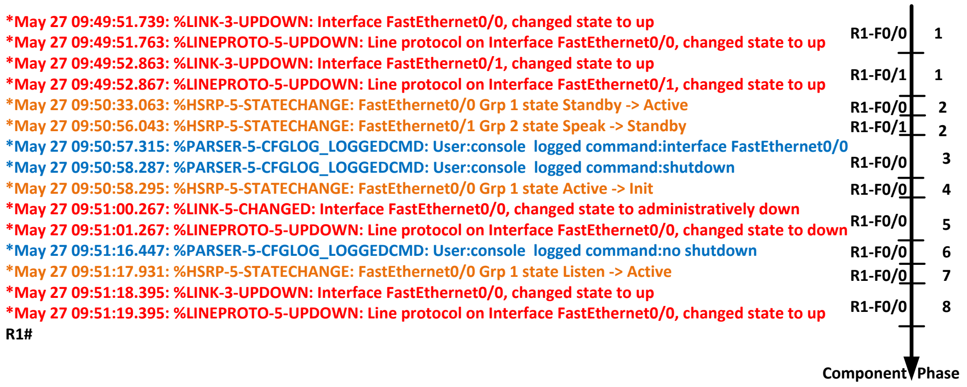
   The *seq no* provides sequential identifiers for log messages (it can be enabled using the command "*service sequence-numbers"* in configuration mode). The *timestamp* is configured using the command *service timestamps log datetime msec* in configuration mode. In this case study, we enable timestamp only. *facility* refers to the system on the device for which we want to set logging (e.g., Kern (Kernel), SNMP, etc.). *Severity* is a single-digit code from 0 to 7 specifying the severity of the message (e.g., 0:emergencies, 1:alerts, 2:critical, 3:errors, 4:warnings, 5:notifications, 6:informational, 7:debugging). *MNEMONIC* is a text string that uniquely describes the message. *description* is a text string containing detailed information about the event being reported.
2. User actions: Cisco IOS stores configuration commands entered by users (e.g., configuring an interface or a protocol) using the config logger. For example, the following log shows that the user has shut down the FastEthernet0/0 interface:

   ```
   *May 27 09:04:37.227: %PARSER-5-CFGLOG_LOGGEDCMD: User:console logged command:interface FastEthernet0/0
   *May 27 09:04:38.475: %PARSER-5-CFGLOG_LOGGEDCMD: User:console logged command:shutdown
   ```

3. Debug messages: They should only be used to troubleshoot specific problems because debugging output is assigned high priority in the CPU process. Hence, it can render the system unusable. The following debug output is produced after enabaling debugging for the HSRP feature (using the command "*debug standby events*"). It illustrates a state change from Speak to Standby on interface Fa0/0 for Group 1:
   *May 24 11:15:41.255: HSRP: Fa0/0 Grp 1 Redundancy "hsrp-Fa0/0-1" state Speak -> Standby

4.4 Log Collection and Segmentation

Figure 6 illustrates the collected log from router R1 (without enabling the sequence number and debugging options). Following the guidelines introduced in Sect. 3, the log has been decomposed into 10 execution phases, where each phase target a single component and describes one and only one type of actions/events. We distinguish two sub-components R1-F0/0 and R1-F0/1, denoting the FastEthernet interfaces within router R1. Phase numbering follows sequential order and provided for each component separately. For instance, the first phase, named R1-F0/0, in Fig 6 illustrates system log messages describing the state of the interface FastEthernet0/0, while the second phase of R1-F0/0 component describes an HSRP state change (i.e., %HSRP-5-STATECHANGE) from Standby to Active. Phase 3 of R1-F0/0 shows that the user has entered the config mode and shut down the interface F0/0.



```
*May 27 09:49:51.739: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
*May 27 09:49:51.763: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
*May 27 09:49:52.863: %LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to up
*May 27 09:49:52.867: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to up
*May 27 09:50:33.063: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Standby -> Active
*May 27 09:50:56.043: %HSRP-5-STATECHANGE: FastEthernet0/1 Grp 2 state Speak -> Standby
*May 27 09:50:57.315: %PARSER-5-CFGLOG_LOGGEDCMD: User:console  logged command:interface FastEthernet0/0
*May 27 09:50:58.287: %PARSER-5-CFGLOG_LOGGEDCMD: User:console  logged command:shutdown
*May 27 09:50:58.295: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Active -> Init
*May 27 09:51:00.267: %LINK-5-CHANGED: Interface FastEthernet0/0, changed state to administratively down
*May 27 09:51:01.267: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to down
*May 27 09:51:16.447: %PARSER-5-CFGLOG_LOGGEDCMD: User:console  logged command:no shutdown
*May 27 09:51:17.931: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Listen -> Active
*May 27 09:51:18.395: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
*May 27 09:51:19.395: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R1#
```

Fig. 6. Log from Router R1

Next, correlations between the extracted execution phases are identified. In our context, exception path triggering events/actions/errors include interface state changes (e.g., up or down) and the administrator shutting/unshutting down interfaces. Events involving HSRP state changes are considered to be part of exception paths since they are supposed to implement fault recovery mechanism.

For example, shutting down the interface F0/0 in phase 3 of R1-F0/0 have triggered an HSRP state change moving the protocol state from *Active* to *Init*. Finally, the correlations between the execution phases are mapped to the UCM notation as shown in Fig. 7(a). Figures 7(b), 7(c), 7(d), 7(e), and 7(f) illustrate examples of UCM plugins corresponding to some execution phases stubs. Figure 7(g) illustrates metadata attributes relative to the responsibility *HSRP-STATECHANGE-F0/0-Grp1-Listen-Active*, while Fig. 7(h) illustrates the metadata attributes relative to the subcomponent F0/0.

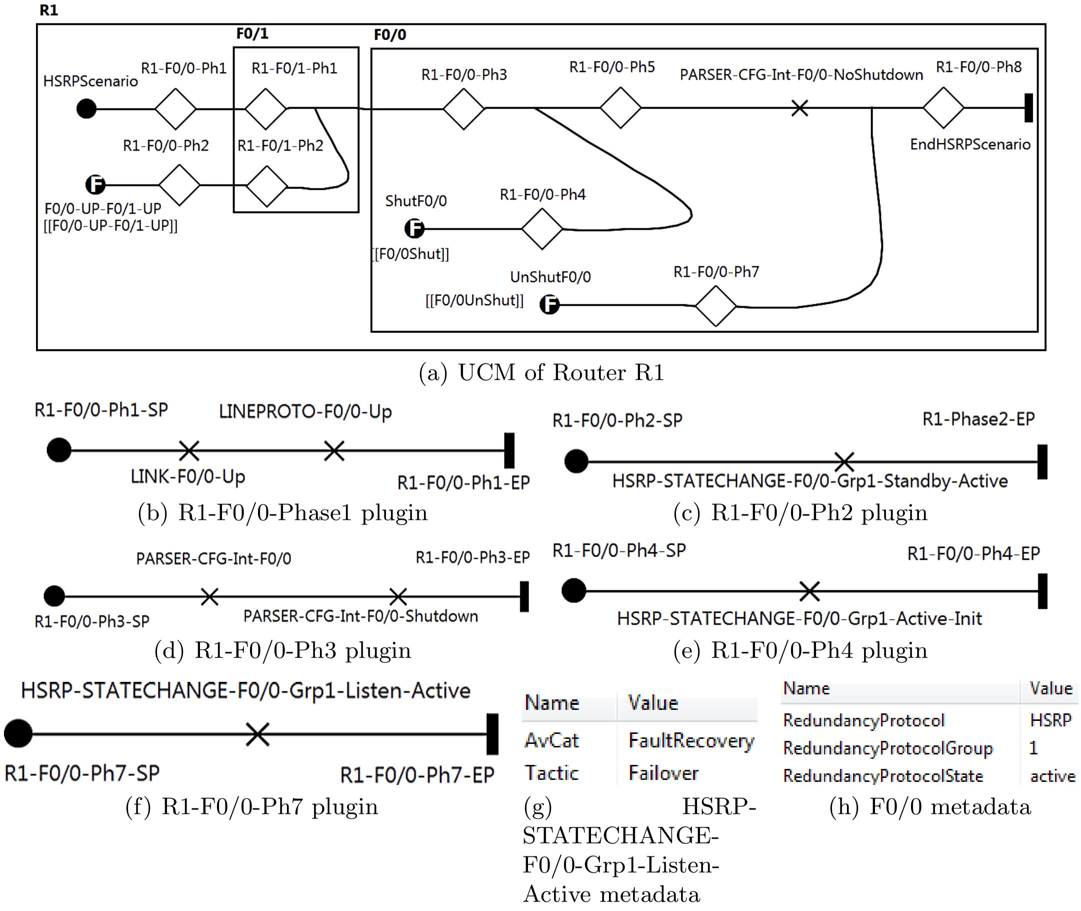
To demonstrate the applicability of our approach in the presence of more than one system log, we have captured the log from router R2. Figure 8 illustrates the merged log for routers R1 and R2. Sixteen phases, involving 4 subcomponents, have been identified. The resulting UCM is depicted in Fig. 9.

It is worth noting that our choice to consider the entire log without neither customization nor chopping some parts is two fold. First, we would like to demonstrate the UCM visualization of more than one subcomponent. Second, although the scenario focuses on the HSRP group 1, it is important to show that actions/events related to this group do not impact group 2 (i.e., absence of feature interactions).

5 Discussion and Threats to Validity

One important objective of this research is to capture non-functional requirements from system execution traces. Our approach uses the high level requirement description language Use Case Maps to describe visually and using metadata availability requirements. UCMs offers a flexible way to represent such requirements at different levels of abstractions using the stub/plugin concept. However, our proposed approach and the experimental case study are subject to several limitations and threats to validity, categorized here according to three important types of threats identified by Wright et al. [23].

Regarding *internal validity*, it might not be sufficient to establish accurate correlations between execution phases without additional semantic information about the running system. For example, in our case study, the log entries corresponding to stubs R1-F0/0-Ph2 and R1-F0/1-Ph2 take place after both interfaces F0/0 and F0/1 came up (Fig. 7(a)). Although, these two events represent the triggers for the R1-F0/0-Ph2 (HSRP group 1) and R1-F0/1-Ph2 (HSRP group 2) phases, we cannot refine such correlation with the available information at hand (i.e., triggers and exception path events). Actually, R1-F0/0-Ph2 and R1-F0/1-Ph2 should be trigged by R1-F0/0-Ph1 and R1-F0/1-Ph1, respectively. Additional, semantic rules are needed in order to achieve accurate correlations. Another possible risk is log complexity. In the presented case study, we have used routers with simple configuration and limited set of configured interfaces. In production networks dozens of features and protocols are configured and interacts with each other. This issue can be mitigated by applying log customization based on a thorough understanding of the deployed protocols and their possible interactions.



(a) UCM of Router R1

(b) R1-F0/0-Phase1 plugin

(c) R1-F0/0-Ph2 plugin

(d) R1-F0/0-Ph3 plugin

(e) R1-F0/0-Ph4 plugin

(f) R1-F0/0-Ph7 plugin

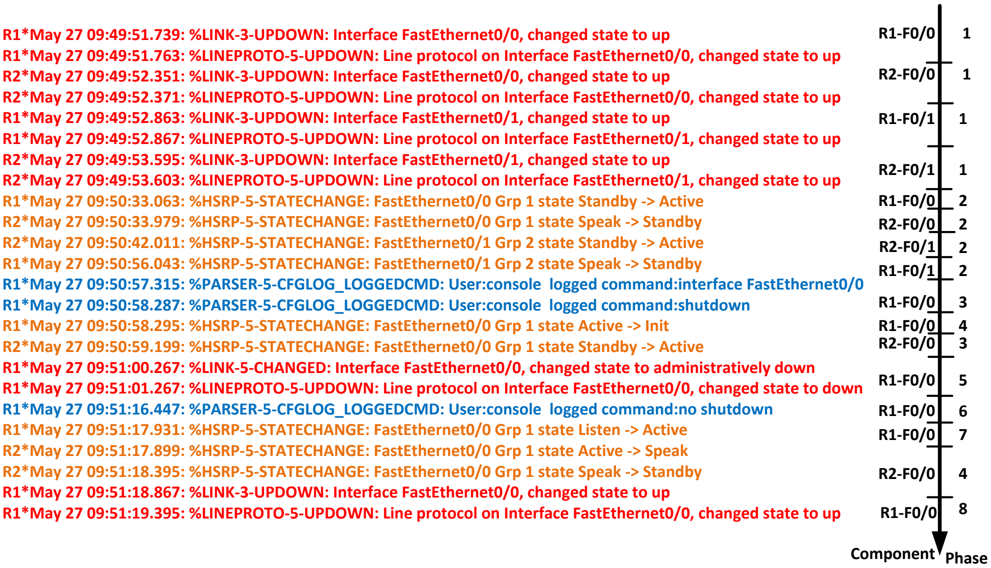
Name	Value
AvCat	FaultRecovery
Tactic	Failover

(g) HSRP-STATECHANGE-F0/0-Grp1-Listen-Active metadata

Name	Value
RedundancyProtocol	HSRP
RedundancyProtocolGroup	1
RedundancyProtocolState	active

(h) F0/0 metadata

Fig. 7. R1 UCM and its related plugins and metadata attributes



```
R1*May 27 09:49:51.739: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
R1*May 27 09:49:51.763: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R2*May 27 09:49:52.351: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
R2*May 27 09:49:52.371: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R1*May 27 09:49:52.863: %LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to up
R1*May 27 09:49:52.867: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to up
R2*May 27 09:49:53.595: %LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to up
R2*May 27 09:49:53.603: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to up
R1*May 27 09:50:33.063: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Standby -> Active
R2*May 27 09:50:33.979: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Speak -> Standby
R2*May 27 09:50:42.011: %HSRP-5-STATECHANGE: FastEthernet0/1 Grp 2 state Standby -> Active
R1*May 27 09:50:56.043: %HSRP-5-STATECHANGE: FastEthernet0/1 Grp 2 state Speak -> Standby
R1*May 27 09:50:57.315: %PARSER-5-CFGLOG_LOGGEDCMD: User:console  logged command:interface FastEthernet0/0
R1*May 27 09:50:58.287: %PARSER-5-CFGLOG_LOGGEDCMD: User:console  logged command:shutdown
R1*May 27 09:50:58.295: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Active -> Init
R2*May 27 09:50:59.199: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Standby -> Active
R1*May 27 09:51:00.267: %LINK-5-CHANGED: Interface FastEthernet0/0, changed state to administratively down
R1*May 27 09:51:01.267: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to down
R1*May 27 09:51:16.447: %PARSER-5-CFGLOG_LOGGEDCMD: User:console  logged command:no shutdown
R1*May 27 09:51:17.931: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Listen -> Active
R2*May 27 09:51:17.899: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Active -> Speak
R2*May 27 09:51:18.395: %HSRP-5-STATECHANGE: FastEthernet0/0 Grp 1 state Speak -> Standby
R1*May 27 09:51:18.867: %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
R1*May 27 09:51:19.395: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
```

Fig. 8. Resulting Log from Routers R1 and R2

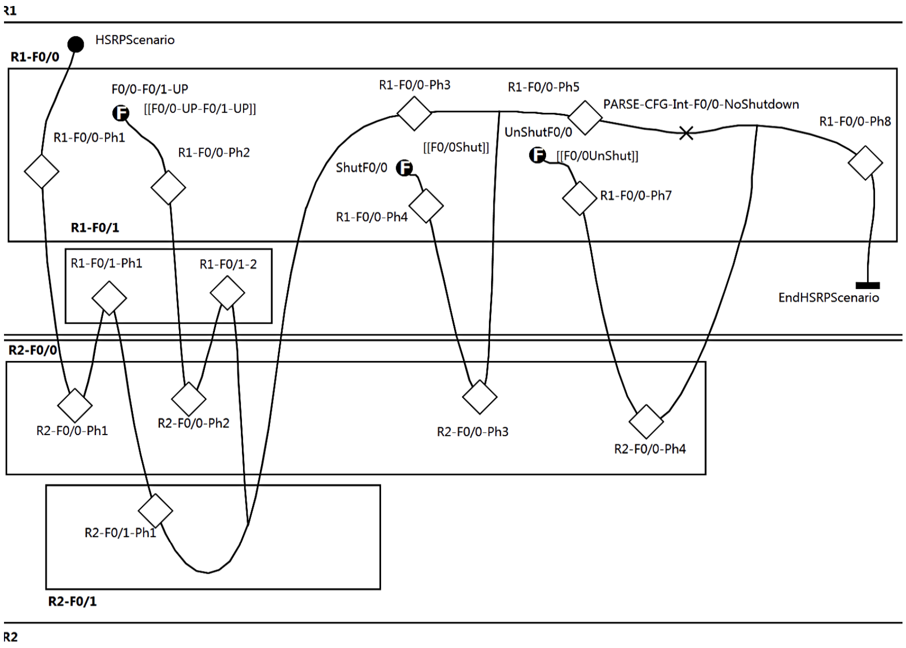


Fig. 9. UCM visualization of the recovery of the merged logs from routers R1 and R2

In terms of *external validity*, there is some risk related to merging logs coming from different devices. This issue is more serious when we deal with equipments from different vendors. Indeed, depending on the devise type and its configuration, discrepancies may arise in terms of the time reference and the event logger priority (e.g., an event logger can have a low priority on one device and a high priority on another), which may lead to a merged log with incorrect chronological order of events. The time reference issue can be mitigated by using the NTP (Network Time Protocol) protocol, which allows for clock synchronization between computer systems over packet-switched, variable-latency data networks.

As for the *construct validity*, scalability represents the most important limitation. As logs becomes more complex, the number of phases becomes difficult to manage and hence, difficult to visualize and to navigate through. Although, the UCM language offers a good encapsulation mechanism through the stub/plugin concept, models can rapidly become messy with overlapping paths and components. However, log customization (e.g., using abstraction techniques) and reduction (e.g., reduce the time stamp window) may help reduce the severity of the scalability issue.

6 Conclusions and Future Work

In this paper, we have proposed a novel UCM-based approach to recover and visualize availability requirements from execution traces. To this end, our proposed approach is built upon previous extensions of the UCM language with availability annotations covering the well-known availability tactics by Bass et al. [15]. Logs from various interacting components can be merged, customized, then segmented into execution phases. The resulting execution stages are then visualized using a combination of UCM regular and exception paths bound to the set of interacting components. Metadata of responsibilities and components implementing fault detection and recovery tactics are captured in an integrated UCM view.

As a future work, we aim at automating the proposed approach. Furthermore, we plan to investigate the design of semantic rules to better correlate the different execution phases. This would allow for more accurate UCM availability models.

Acknowledgment

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at King Fahd University of Petroleum & Minerals (KFUPM) for funding this work through project No. IN131031.

References

1. Corbi, T.A.: Program understanding: Challenge for the 1990s. IBM Systems Journal **28**(2) (1989) 294–306

2. Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L., Koschke, R.: A systematic survey of program comprehension through dynamic analysis. IEEE Transactions on Software Engineering **35**(5) (2009) 684–702
3. Zaidman, A.: Scalability solutions for program comprehension through dynamic analysis. In: Proceedings of the Conference on Software Maintenance and Reengineering. CSMR '06, Washington, DC, USA, IEEE Computer Society (2006) 327–330
4. Reiss, S.P.: Visualizing program execution using user abstractions. In: Proceedings of the 2006 ACM Symposium on Software Visualization. SoftVis '06, New York, NY, USA, ACM (2006) 125–134
5. Koskimies, K., Mössenböck, H.: Scene: Using scenario diagrams and active text for illustrating object-oriented programs. In: Proceedings of the 18th International Conference on Software Engineering. ICSE '96, Washington, DC, USA, IEEE Computer Society (1996) 366–375
6. Walker, R.J., Murphy, G.C., Freeman-Benson, B., Wright, D., Swanson, D., Isaak, J.: Visualizing dynamic software system information through high-level models. In: Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. OOPSLA '98, New York, NY, USA, ACM (1998) 271–283
7. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. **1**(1) (January 2004) 11–33
8. Jalote, P.: Fault Tolerance in Distributed Systems. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1994)
9. ITU-T: Recommendation Z.151 (10/12), User Requirements Notation (URN) language definition, Geneva, Switzerland (2012)
10. Hassine, J., Hamou-Lhadj, A.: Towards the generation of AMF configurations from use case maps based availability requirements. In Khendek, F., Toeroe, M., Gherbi, A., Reed, R., eds.: SDL 2013: Model-Driven Dependability Engineering. Volume 7916 of Lecture Notes in Computer Science. (2013) 36–53
11. Hassine, J., Mussbacher, G., Braun, E., Alhaj, M.: Modeling early availability requirements using aspect-oriented use case maps. In Khendek, F., Toeroe, M., Gherbi, A., Reed, R., eds.: SDL 2013: Model-Driven Dependability Engineering. Volume 7916 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 54–71
12. Hassine, J., Gherbi, A.: Exploring early availability requirements using use case maps. In Ober, I., Ober, I., eds.: SDL Forum. Volume 7083 of Lecture Notes in Computer Science., Springer (2011) 54–68
13. Hassine, J.: Early availability requirements modeling using use case maps. In: ITNG, IEEE Computer Society (2011) 754–759
14. Hassine, J.: Describing and assessing availability requirements in the early stages of system development. Software & Systems Modeling (2013) 1–25
15. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
16. jUCMNav v5.5.0: jUCMNav Project, v5.5.0 (tool, documentation, and metamodel). http://jucmnav.softwareengineering.ca/jucmnav (2014) Last accessed, June 2014.
17. Cisco Systems: Hot Standby Router Protocol Features and Functionality. http://www.cisco.com/c/en/us/support/docs/ip/hot-standby-router-protocol-hsrp/9234-hsrpguidetoc.pdf (2006) Online; accessed 27-May-2014.

18. Hassine, J., Rilling, J., Dssouli, R.: Timed Use Case Maps. In: System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, May 31 - June 2, 2006, Revised Selected Papers. (2006) 99–114
19. Li, T., Cole, B., Morton, P., Li, D.: Cisco Hot Standby Router Protocol (HSRP). RFC 2281 (Informational) (March 1998)
20. Nadas, S.: Virtual router redundancy protocol (vrrp) version 3 for ipv4 and ipv6. RFC 5798 (Proposed Standard) (mar 2010)
21. GNS3: Graphical network simulator, gns3 v0.8.6 (2014)
22. Cisco Systems: Internetworking Technologies Handbook. Cisco Press networking technology series. Cisco Press (2004)
23. Wright, H.K., Kim, M., Perry, D.E.: Validity concerns in software engineering research. In Roman, G.C., Sullivan, K.J., eds.: FoSER, ACM (2010) 411–414