UPGRADE CAMPAIGN SIMULATION AND EVALUATION FOR HIGHLY AVAILABLE SYSTEMS

Oussama Jebbar
Engineering and computer science
Concordia University
1455 De Maisonneuve W.
Montreal, QC, Canada
ojebbar@encs.concordia.ca

Ferhat Khendek
Engineering and computer science
Concordia University
1455 De Maisonneuve W.
Montreal, QC, Canada
ferhat.khendek@concordia.ca

Maria Toeroe
Ericsson Canada Inc.
8275 route Transcanadienne
Saint-Laurent, QC, Canada
maria.toeroe@ericsson.com

ABSTRACT

High Availability (HA) is an important nonfunctional requirement of carrier grade services. These services must not experience more than five minutes of downtime per year including outage due to planned upgrades. The Service Availability Forum (SAF) developed a set of standard services for an HA enabling middleware. Among these services, the Availability Management Framework (AMF) is responsible for managing application/service availability, while the Software Management Framework (SMF) is in charge of orchestrating upgrade campaigns that perform changes on these applications. Upgrades can be performed in different ways and may have different characteristics, such as induced service outage or upgrade duration. We propose an approach which is based on the Discrete Event System Specification (DEVS) formalism for upgrade campaign simulation, evaluation and selection using the concepts of best case and worst case scenarios.

Keywords: High Availability, Service Availability Forum, Software Management, Upgrade Campaign Execution Time, Service Outage.

1 INTRODUCTION

An upgrade campaign is the process of migrating a system from its current configuration to a target configuration that, for example, deploys a new version of some software. In a Service Availability Forum (SAF) (SAF 1) compliant environment, the Software Management Framework (SMF) (SAF 3 2011) orchestrates this migration by following a roadmap, an upgrade campaign specification, which describes the changes to do and their ordering. An upgrade campaign specification describes the actions to initialize and to wrap up the campaign, and the upgrade procedures that compose the body of the campaign. Each procedure has an upgrade scope, an upgrade method, and the common attributes of the steps composing this procedure. At execution time SMF decomposes each procedure into a number of upgrade steps based on the upgrade scope, the upgrade method and the current system configuration. Each step has a sequence

SpringSim-TMS/DEVS 2017, April 23-26, Virginia Beach, VA, USA

of actions. Some of these actions are performed online while the service is being provided, and others are performed offline by taking a subset of the service providers out of service to upgrade them. The later may induce some service outage.

Several upgrade campaign specifications might be applicable to take a system from a source configuration to a target one. These campaign specifications have in common the set of changes they have to perform, but they vary in other aspects such as the number of procedures, the ordering of the procedures, or even the choice of the scope and the upgrade method of each procedure. Therefore, some upgrade campaign specifications may induce more outage or may take longer time to execute than others. In order to select one upgrade campaign over another, one has to consider every behavior that may take place during its execution. This includes the behavior of the SMF executing the upgrade steps, the upgrade procedures and the upgrade campaign on one hand; and the behavior of the Availability Management Framework (AMF) (SAF 2 2011) with respect to the system entities represented by the configuration objects which are the targets of the upgrade campaign and whose availability is managed by AMF on the other hand. Thus, making this decision is not straightforward.

In this paper we propose a simulation based approach for the evaluation of upgrade campaign specifications. Our approach is based on the Discrete EVent System Specification (DEVS) formalism (Bernard et al. 2000). We extend DEVS for the modeling of upgrade campaign specifications and their execution environment. We also extend the DEVS-Suite simulator (DEVS-Suite). The simulation of an upgrade campaign may randomly take many different scenarios inducing different outages and resulting in different execution times. To evaluate and compare effectively upgrade campaign specifications from the perspectives of both induced outage and execution time, we define the concepts of best case and worst case scenarios and we target the simulation of these specific scenarios for each upgrade campaign specification. We propose an upgrade campaign specification selection/elimination process based on this evaluation. This process selects a set of applicable upgrade campaign specifications according to the constraints of acceptable outage time and maintenance window. The final selection among the applicable upgrade campaign specifications is left to the system administrator who can favor one criteria over the other knowing the specificities of the system under consideration.

The rest of this paper is organized as follows: Section 2 provides the necessary background on the SAF standards and DEVS formalism. In Section 3 we describe our overall approach and the different DEVS models involved. We introduce the best case and worst case scenarios and the different aspects of our simulation in Section 4. In Section 5 we elaborate on the selection/elimination of upgrade campaign specifications. We discuss related work in Section 6 before concluding in Section 7.

2 BACKGROUND

2.1 SAF Systems

High Availability (HA) is an important requirement for carrier grade services. As mentioned earlier SAF (SAF 1) has defined a set of middleware services to enable the development of commercial-of-the-shelf (COTS) components for HA systems. Among these services AMF and SMF play important roles.

AMF (SAF 2 2011) manages the service provider resources under its control to ensure the availability of the services they provide. To achieve this, AMF requires a configuration that describes the entities composing the system and their relations. The smallest building block of this configuration is the component which represents a resource such as a running instance of some software. Each component can provide one or more basic services called Component Service Instance (CSI). A set of components form a Service Unit (SU) to provide a Service Instance (SI), which is an aggregation of CSIs provided by the components of this SU. To ensure service availability, redundant SUs that can provide the same SIs are grouped into a Service Group (SG). To provide the services an SI represents, at runtime AMF assigns the SI to some SUs of the protecting SG. If an assigned SU is unable to provide the service, AMF instructs another SU of the same SG to take over. An SG is usually deployed on a subset of the nodes composing the

cluster. The runtime state of the system is captured through a set of attributes representing among others the different phases of the component life cycle (presence state), the eligibility of an SU to provide the service (administrative state), and SI provisioning levels (assignment state). The state of every entity in the configuration impacts the state of its parent and vice-versa. These states follow the state model described with Finite State Machines (FSM) in the SAF AMF specification (SAF 2 2011). The most relevant state for this paper is the SI assignment state, which can be either fully assigned, partially assigned, or unassigned. A service is considered to be interrupted when the representing SI is in the unassigned assignment state. In the rest of this paper we refer by a SAF system to a system whose service availability is managed by an implementation of the SAF AMF specification (SAF 2 2011).

For managing the software in a SAF system, SMF (SAF 3 2011, M. Toeroe and F. Tam 2012) has been defined and it covers two parts, the software delivery and the software deployment. The software delivery describes how software bundles are delivered to a SAF system. Every software bundle should be accompanied by the Entity Types File (ETF) (SAF 3 2011) that serves as a manifest describing whatever component and other entity types the bundle delivers and the service types they can provide. ETF files are usually provided by the software vendor, and contain data about the deployment of components of a given type. The software deployment part describes how an upgrade campaign should be specified and how it is executed to deploy a set of configuration changes. In SAF systems, these changes are performed by an SMF engine based on the content of an upgrade campaign specification file. The changes are distributed over a set of partially ordered procedures. Each procedure section specifies an upgrade method (rolling upgrade or single step upgrade), and the upgrade scope which consists of the set of configured objects representing entities in the system on which the procedure will act. When an SMF engine is given such a file, it executes the upgrade campaign and the specified upgrade procedures as they apply to the current state of the system. More specifically, each procedure is divided into upgrade steps depending on its upgrade method and as its scope maps to the configuration of the SAF system. The behavior associated with the upgrade steps, upgrade procedures and upgrade campaigns are described using FSMs in (SAF 3 2011). They are represented in the SMF information model as upgrade objects. To realize the changes, the SMF engine performs some upgrade actions, such as software installation/removal, interactions with AMF and configuration modifications. As part of this SMF may order AMF to lock and restart a set of components, a set of SUs, or even a set of nodes depending on the upgrade scope of the upgrade step in question. These actions are the main causes of service outage induced by an upgrade campaign. The SMF specification remains at a high level of abstraction, and some decisions are left open for the implemention.

2.2 DEVS Formalism

The Discrete EVent System Specification (DEVS) is a formalism proposed by Bernard P. Zeigler (Bernard et al. 2000). It is commonly used for discrete event system modeling and analysis. A DEVS model can be atomic to capture the behavior of a component or coupled to capture the structure of a system and the components composing it. A system in DEVS can interact with its environment through input and output events, leading to a transition from one state to another. A DEVS model may also generate an output/state change without any interaction with its environment when time spent in a state has elapsed as per the definition of the system. Several simulators have been built for DEVS, in this paper we use DEVSJAVA, which is part of the DEVS-Suite project (DEVS-Suite), led by Arizona State University. The rationale behind the choice of this formalism and simulator is that:1) DEVS formalism can model both synchronous and asynchronous communications that are used in SAF systems, and in particular during upgrade campaigns execution; 2) DEVS-Suite simulator allows for tracking the state of a model, which is necessary for our analysis (to trace the outage and execution time); and 3) DEVS-Suite simulator offers a reduced visual complexity compared to other simulators such as Ptolemy (Sungung et al. 2009). This simplifies the visualization of the HA-system regardless of its size.

3 OVERALL APPROACH

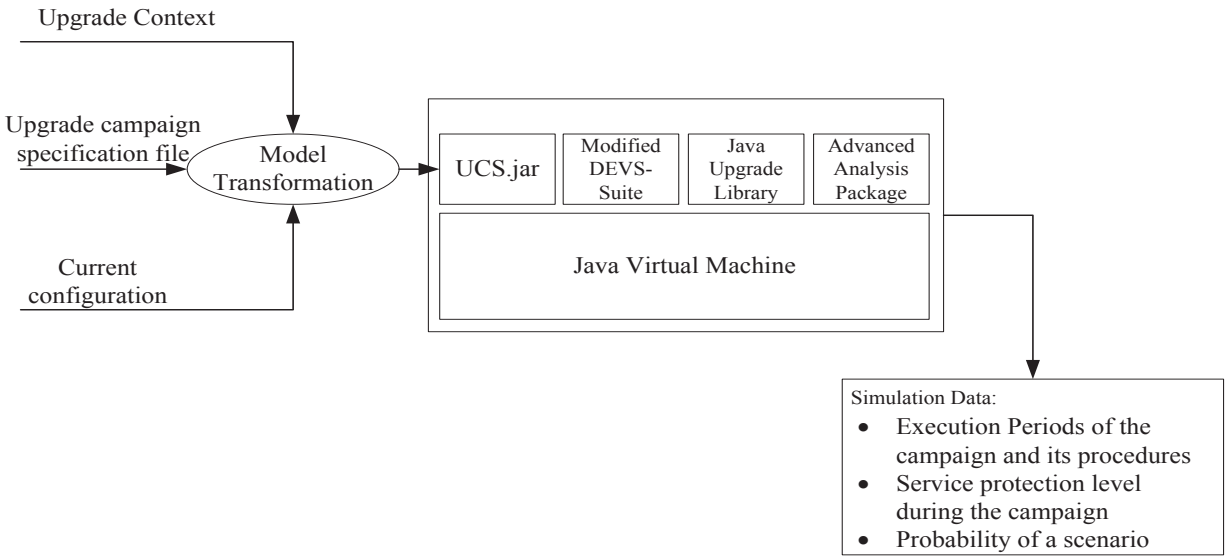


Figure 1: Simulation environment and information flow.

As shown in Figure 1, our approach for the simulation and evaluation of upgrade campaign specifications takes three inputs. The upgrade campaign specification file describes the campaign to be executed. This file will allow our simulation environment to decompose properly the procedures on one hand, and run the appropriate behavior for each instantiated upgrade object based on its specified attributes on the other hand. The system configuration of the SAF system to be upgraded provides the information required to properly instantiate and simulate the configuration objects. It also describes how these objects relate to each other. These relations can be either dependencies between services or lifecycle dependencies between components as defined in the AMF specification (SAF 2 2011). Finally, the upgrade context includes the set of ETF files describing the upper and lower bounds for the durations of operations of the component types used in the configuration. For the simulation, we need additional information such as the failure rates of the component types and their lifecycle operations, the failure rates of software operations (installation/removal), as well as the durations of node startup and shutdown operations.

From the aforementioned input using a model transformation we create a Java class for a coupled DEVS model that contains a description of the upgrade campaign's runtime environment. This Java class is packaged into a jar (UCS.jar) and loaded into the simulation environment. One of the main components of the simulation environment is the Java Upgrade library which defines the structure and the behavior of the DEVS atomic models used to simulate the configuration and upgrade objects involved in the campaign execution and the transformation instantiates and combines into a DEVS coupled model (UCS.java). The actual behavior that drives the simulation is implemented at the level of every concept defined in this library, and it implements the description provided in the SAF standards (SAF 3 2011, SAF 2 2011). As the UCS.java instantiates the classes defined in the Java Upgrade library to describe the simulated campaign, it should also be loaded into a simulator for the simulation. For this purpose we use a Modified-DEVS-Suite, a version of the DEVS-Suite simulator (DEVS-Suite) extended for our needs. The extensions to DEVS formalism and DEVS-Suite simulator are described in the next subsection. Finally, the best case and worst case scenarios, as defined in this work and described in details in Section 4, are implemented in the Advanced Analysis Package. This package is used to control the simulation scenario in order to force it to take a predefined execution path (best or worse) rather than taking a random execution path based on the simulated behaviors and their characteristics. This control is applied on every action during the campaign's execution, and can be either a control over the duration the action will take, a control over when the action will start, or a control over when an action should fail. Using the simulation one can have an insight into the following aspects of the campaign:

- The execution time of the campaign and each of its procedures, which can help completing/revising the upgrade campaign specification by filling in some missing attributes as well as comparing upgrade campaign specifications from the point of view of their execution time.
- The SIs assignment states during the execution of the campaign, which can help completing/revising the specification of upgrade procedures, and also evaluating upgrade campaign specifications from the perspective of the induced service outage.
- Probability of the scenario, which can be used to obtain the probability of the best and worst case scenarios.

3.1 Extensions to DEVS Formalism and Simulation Engine

The DEVS formalism, as it is, can only model choreographically performed collaborations, meaning that the logic of the collaboration is distributed among collaborating entities. In our case, some of the collaborations are orchestrated, meaning that there is a central entity orchestrating the collaboration between a set of collaborating entities. If we take AMF, for example, at any point in time it may have different services to switchover to different entities with different time offsets for the next event. To make the orchestration of these events easier, the atomic DEVS model representing AMF should have a state-independent time-awareness. To capture this kind of collaborations we extended the DEVS time function in a way that not only the states have life spans but also the events. Therefore, and unlike a usual DEVS model that is considered imminent (or ready for a transition) only when it receives an incoming event or the life span of its current state expires, a DEVS model simulated under this extension will also be considered imminent when the lifespan of one of its events expires. The DEVS model responds to this transition trigger by firing that event to its destination and going back to wait for the next transition trigger.

We extended the DEVS-Suite simulator APIs to handle output events with lifespan as argument. Keeping in mind backward compatibility, we defined zero (0) as the default lifespan value for an event that should be issued right after its creation. We also extended the simulation mechanisms (simulators and coordinators) to include the event lifespan in the calculation of the next iteration time.

3.2 Extensions to SAF Specifications

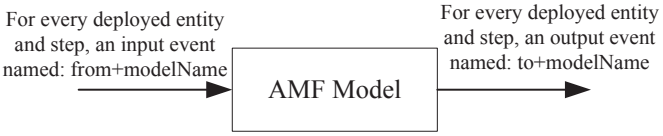
The information provided in the configuration and ETF files are not enough to perform an accurate simulation. The failure rates and some execution times are not in the standard specifications. We added the information needed for our simulation as part of the Upgrade Context. For instance, we added attributes for a software bundle to describe the failure model of software operations and their default execution time. Similarly, we added attributes to the component type to express its failure model and the failure models of its lifecycle operations. Note that the time attributes in our model are bounded times, i.e. they provide a lower bound and an upper bound for the execution of the given operation. These upper and lower bounds are used to simulate the worst case and the best case execution scenarios. The set of vendor provided ETFs (which are basically xml files), in addition to the durations of startup and shutdown of each node (provided as user input) compose what we refer to as upgrade context.

3.3 Transformation

The model transformation shown in Figure 1 generates a DEVS coupled model (UCS.java) from the given configuration, upgrade context and upgrade campaign specification files according to the mapping described in Table 1. The AMF model is first added to the UCS.java, then we model the configuration by adding an AMF Entity DEVS model for each component, CSI, SU, SI, SG, and node in the configuration and their respective couplings to the AMF Model. The upgrade campaign, on the other hand, is represented by an UpgradeCampain model and coupling it to the UpgradeProcedure models corresponding to each specified upgrade procedure in the upgrade campaign specification file. Finally, based on the specified scope and upgrade method, we add the UpgradeStep models that represent the steps of each procedure and we establish the couplings to the AMF Model and the UpgradeProcedure models accordingly.

The DEVS atomic models defined in the Java Upgrade Library are designed according to the communication patterns between the different objects involved in the execution of an upgrade campaign of a given system. The upgrade campaign object, for instance, needs to communicate with the administrator and its procedures, which explains the I/O ports associated with its DEVS model. Similarly, an upgrade procedure communicates with the campaign and the associated steps. Upgrade steps exchange events with the procedure and with the AMF to perform actions on the logical entities in the AMF configuration. Thus, there is a need for an atomic DEVS model representing AMF. It was designed with an input event from and an output event to every upgrade step and to every configuration object in the final DEVS model. The main responsibility of the atomic DEVS model representing AMF is the interpretation of the administrative operations issued by upgrade steps on configuration objects and their decomposition into the associated time constrained AMF component management operations as defined in (SAF 2 2011) and for which the time constraints are given in the system configuration. The transformation is written in Epsilon Generation Language (EGL) (Louis et al. 2008) a model transformation language used for model to text transformations.

Table 1: Objects involved in the simulation and their associated DEVS models.

	DEVS Models
Upgrade campaign	
Upgrade procedure	
Upgrade step	
AMF entity	
AMF	

4 BEST CASE AND WORST CASE SCENARIOS

The simulation of an upgrade campaign can give some insight into how an upgrade campaign specification applies to a given configuration, but it remains inconclusive when it comes to the comparison of upgrade campaign specifications. Indeed, the distribution of failures and execution times induces significant randomness into the simulated behavior and makes the simulation results arbitrary, thus, unreliable for the purposes of comparison. Depending on the scenarios that are randomly followed, one upgrade campaign specification can give better results than another, but it is unclear whether this is because of the difference in the design, or because the random scenarios only exposed a more advantageous execution path for one upgrade campaign specification than for the other one. We need to evaluate and compare the upgrade campaign specifications under similar circumstances with similar scenarios; therefore we consider two

particular cases: the best case and the worst case scenarios. To define the best case and worst case scenarios, we take into consideration three aspects related to the campaign simulation/execution:

- Software operations: The execution of a software operation may succeed or fail and does not always take the same duration. The upgrade context previously described captures this fact by associating with every software operation a failure rate and a bounded duration attribute specifying a lower bound and an upper bound. However, the upgrade campaign specification may constrain these operations by a default timeout, which effectively replaces the upper bound value as SMF engages the appropriate upgrade repair mechanism if the timer expires.
- Upgrade repair mechanisms: Upgrade actions are subject to failure, and depending on the failure stage SMF can engage different chains of actions involving both software and component management operations. Moreover, the upgrade campaign specification specifies for each step the maximum number of attempts before reporting the failure of the upgrade campaign due to the failure of this step. For different stages of failures and different failures' count during the execution of every step in the campaign we can have different scenarios that induce different levels of service disruption and take different durations. We use edge combinations of the two factors in the definition of the best and worst case.
- AMF configuration object behavior: As mentioned previously, upgrade actions that involve configuration objects are decomposed into AMF component management operations. Since each applicable management operations may succeed or fail taking different amounts of time the upgrade context includes a failure rate and a bounded time attribute for each of these management operations. These failure rates and bounded times can either be vendor provided as a result of a benchmarking or experimentations performed on the software before distribution or estimated by the administrator based on historical data. In addition, a given AMF configuration also constrains all these operations by timeouts, meaning that if an operation is taking more time than the configured timeout, AMF assumes the operation has failed and engages in a recovery and repair actions, which map into AMF management operations sometimes at component in other cases at node level. In case of a failure AMF may also reattempt some of the operations for a configured number of times before reporting a failure. All these aspects are used to define the best execution and worst execution of an operation on a given component.

The SMF specification left open some implementation decisions to be made by the SMF engine implementer. Among these, the most relevant decision to our work is whether the SMF engine is capable of parallel execution of the upgrade procedures. As indicated in Section 2.1, the upgrade procedures of an upgrade campaign are only partially ordered and, accordingly, an implementation may execute the unordered sets in parallel or serialize them into a fully ordered execution. According to the SAF SMF standard (SAF 3 2011) the fully ordered execution is mandatory for any SMF engine implementation while the partially ordered execution is optional, i.e. it may or may not be supported. As introduced in Section 3, the best case and worst case scenarios are implemented as a control over actions in the Advanced Analysis Package (Figure 1). Note that among the aforementioned three types of control, the control over the duration and the failure of an action impacts the time an action can take and therefore the time the campaign might take. As opposed to this, the control over the time an action is started induces certain level of interference between various upgrade procedures and therefore impacts both the campaign's time and outage.

4.1 Fully Ordered Execution

In a fully ordered execution an SMF engine executes only one upgrade step at a time regardless of the arrangement of these steps into upgrade procedures. Each upgrade step contains a sequence of upgrade actions, some of which are performed by SMF itself, i.e. the software operations, while others are issued towards AMF and therefore decomposed into sequences of AMF management operations according to the applicable AMF configuration object behavior. Thus, from an execution time perspective we only need to control the duration and the failure of each action that takes place in the upgrade campaign. From an outage perspective, the fully ordered execution also means that the upgrade actions cannot interfere with each other and cause more outage. As a result for the fully ordered execution, the best case and worst case scenarios

do not exercise any control over when an action is started, and thus they differ only on the control they exercise over the duration and failure of each action. The respective controls can be described as:

- Best case: each action in the campaign execution is simulated as taking as long as its best execution time, which is described as: 1) the action succeeds at the first attempt; and 2) the action takes the minimum time needed, which is equal to the lower bound value specified for the operation in the upgrade context.
- Worst case: actions may fail and can be retried. Therefore, we also need to consider the SMF upgrade repair mechanisms as well as the AMF recovery and repair actions. We consider these aspects while the execution of the upgrade campaign is still successful. Accordingly, from the upgrade repair mechanisms we take into account the upgrade step retry option only. With respect to the AMF recovery and repair actions as we mentioned they map to AMF management actions and can be handled at the action level. With these considerations in mind we define the worst case execution time at two levels of granularity. First, at upgrade action level (whether it is an AMF action or an SMF action), the worst case execution time is described as: 1) the action succeeds at the last attempt; and 2) the action takes the maximum time allowed which is the applicable timeout in the AMF configuration if it is an AMF action, or the timeout specified in the upgrade campaign specification otherwise. Second, at the upgrade step level where: 1) all the upgrade actions of an upgrade step take as long as their respective worst case execution time as defined in the previous level of granularity; and 2) an upgrade step succeeds only on the last attempt of the allowed retries, while all the other attempts fail on the upgrade action with the worst impact.

4.2 Partially Ordered Execution

The partially ordered execution means that some upgrade procedures are not ordered and the SMF engine can execute them in any order including in parallel as there is no synchronization required between these procedures. A parallel execution of some upgrade procedures obviously shortens the execution time of the upgrade campaign. However the lack of synchronization between the upgrade procedures also means that they may interfere with each other: They may take out of service multiple SUs of the same SG beyond what can be tolerated in the given configuration in terms of outage. The SMF engine implementation may be able to avoid such interferences or not. Thus, the Advanced Analysis Package exercises the three types of control as defined previously in order to consider these aspects in the implementation of the best and worst case scenarios. Accordingly, for the partially ordered execution we define the best case scenario at two levels of granularity. First, at the upgrade step level where the upgrade steps for which the combined effect is minimal are executed simultaneously, and all the upgrade actions of an upgrade step take as long as their respective best case execution time. Second, the upgrade action best case execution time is defined as in the case of the fully ordered execution. As opposed to this, the worst case scenario of the partially ordered execution requires the same control as for the fully ordered execution, with the addition of a control over the starting time of some actions. The steps executed simultaneously are the ones for which the combined effect on the system is maximal.

5 SELECTION/ELIMINATION OF UPGRADE CAMPAIGN SPECIFICATIONS

Given a set of upgrade campaign specifications that can take a system from its current configuration to the same target configuration, we can use our simulation approach to evaluate them and decide which ones are applicable considering some targeted acceptable outage and maintenance window. The applicability of the scenarios discussed in the previous section depends on the capability of the SMF engine. Accordingly, if the SMF engine is not capable of parallel execution the scenarios of Section 4.2 do not apply. All SMF engines must be capable of fully ordered upgrade campaign execution. Since our goal is to meet some acceptable outage and maintenance window we evaluate the upgrade campaigns from the perspective whether these goals can be achieved rather than selecting “the best upgrade campaign”. This is because it

is not straightforward how service outage trades for execution time. Our evaluation (summarized in Figure 2) goes along the following lines:

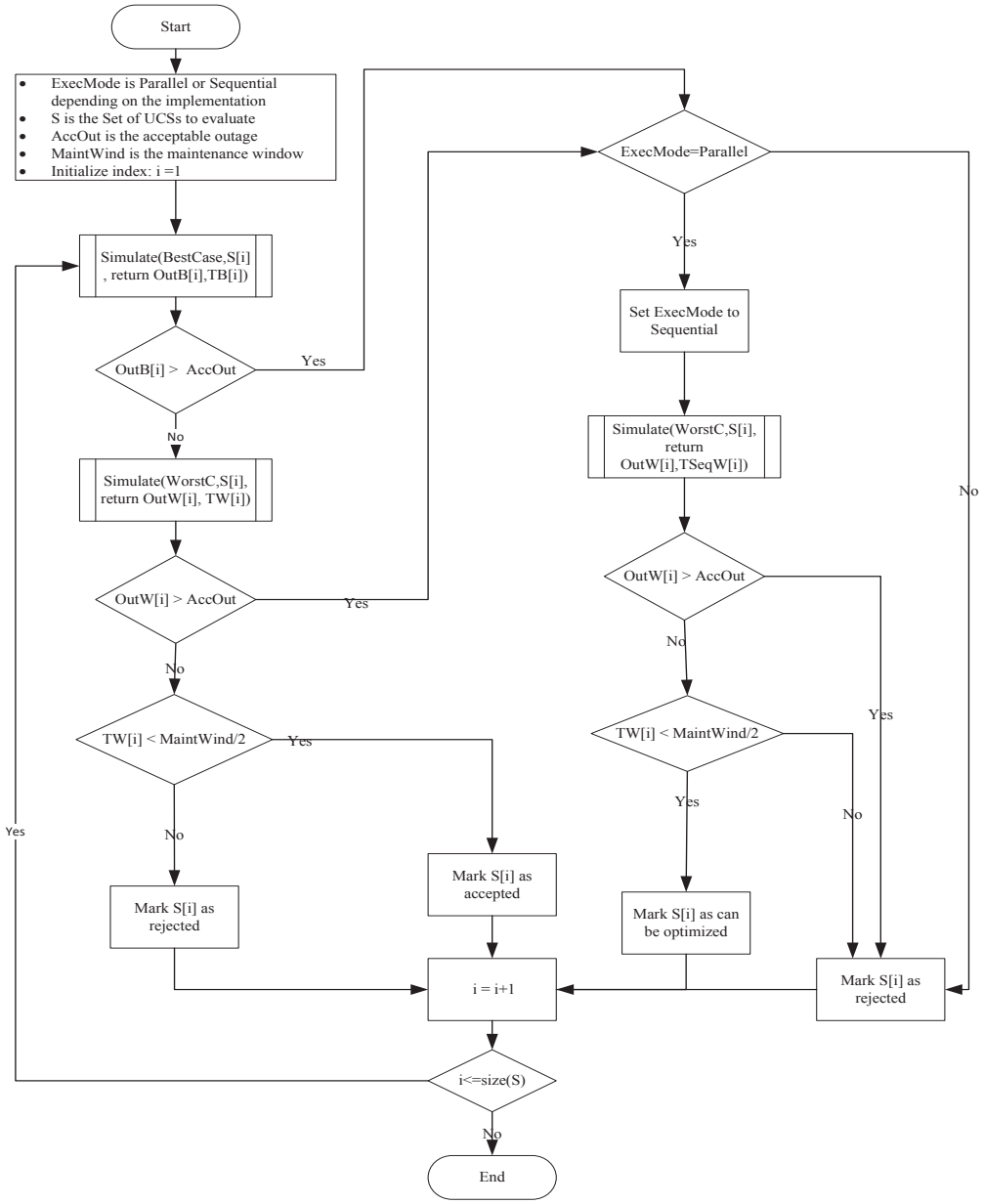


Figure 2: Process of selection/elimination of upgrade campaign specifications based on different criteria.

- First all the upgrade campaign specifications are evaluated for the execution mode applicable to the SMF engine and both the best and the worst case scenarios are evaluated for their execution time and induced outage.
- If the execution mode of the SMF engine is sequential all the upgrade campaign specifications that induce an unacceptable outage for either the best or the worst case scenario can be safely eliminated as there is no guarantee they can meet the outage constraint.
- In the case of parallel execution mode the upgrade campaign specifications violating the outage constraint are further evaluated for the sequential execution as it typically induces less outage. Those upgrade campaign specifications which still result in an unacceptable outage for either

the best or the worst case are eliminated. The remaining upgrade campaign specifications are marked for potential serialization.

- Next the execution times of all the pre-selected upgrade campaign specifications are evaluated with respect to the maintenance window. The main consideration is that we would like to complete the upgrade campaign within half of the available maintenance window. This allows for a graceful rollback of the system to its original configuration should anything go wrong unexpectedly during campaign execution. Accordingly, we eliminate all the upgrade campaigns that result in an execution time greater than the half of the targeted maintenance window. This criterion may be relaxed if a partial or full restoration of the system from a backup is an acceptable recovery and therefore can be used to shorten the rollback time.

The selected upgrade campaign specifications are acceptable albeit some with the need for serialization. They can be compared and further analyzed from the perspective of their induced outages and execution times to pick the one that is the most suitable for the given system and constraints. A system administrator may choose the campaign that takes the least time in order to make better use of the maintenance window, while another one may choose the one that takes the longer time because, for example, it specifies more upgrade steps retries and thereby is more reliable. The choice can also be based on the probabilities associated with the best case and the worst case scenarios and select the one which has the highest probability for the best case scenario. This is also where the tradeoff execution time for service outage and vice versa becomes important. Our selection/elimination process ends with a set of applicable upgrade campaign specifications.

6 RELATED WORK

Upgrade execution time estimate is an ongoing concern among practitioners. Different fora in the Internet are dedicated to this question for specific IBM, Microsoft or Cisco products (Cisco Forum). These discussion groups provide specific solutions/tricks for specific products but do not propose generic solutions. Despite the importance of this problem and a thorough literature search we only found (Michal et al. 2013) and (Xiangqun et al. 2005) as closely related work.

The upgrade execution time for a database has been investigated in (Michal et al. 2013). For some specific types of upgrades, bounded by some constraints on the type of changes to be done on the schema and the data, (Michal et al. 2013) propose an evolutionary algorithm that uses data from previous upgrades to estimate the time required to upgrade every atomic entity in the system (in this case business objects), and then estimate the time required for the next upgrade based on the business objects that will be upgraded and the previously estimated respective upgrade durations. In our work we assume that the time every action takes is provided by, for example, the vendor as these actions are decomposed into operations, which are constrained by the configuration or the upgrade execution. Our work is in the context of SAF, but is not bounded to a specific type of systems, neither limited by the type of changes the system will undergo. Unlike the work presented in (Michal et al. 2013), we also estimate the time of upgrading multiple applications within the same upgrade campaign.

Two types of service outage can affect the availability of a system: unplanned downtime caused by failures and the planned downtime due to upgrades. Therefore, the outage induced by an upgrade campaign is an important factor to consider while estimating the availability of a system. The work in (Xiangqun et al. 2005) presents an assessment of reliability and availability of a Storage Area Network (SAN) after two types of extensions: SONET based SAN extension, or IP based SAN extension. In their analytical models the authors take into consideration system upgrades by setting a fixed number of upgrades per year, their durations and the downtime they induce. Using these models they estimate the availability and reliability of the SAN in the long term. This work is on the estimation of the availability taking into account the downtimes due to upgrades, but not on the upgrade execution time estimation. The work in (Xiangqun et

al. 2005) is along the same lines as the work reported in (Ali et al. 2012, 2013). However, the work in (Ali et al. 2012, 2013) does not consider the effect of upgrade campaigns on the downtime in the long term. The authors propose an approach to estimate the availability of a service provided by a SAF system based on its current configuration. They use Deterministic and Stochastic Petri Nets (DSPNs) (Ajmone Marsan, M., Chiola, G. 1987) to model the configuration, and they explore components' failure modes and recovery actions mapped to every failure mode. They extend the SAF models in a similar way as we do in order to include the missing time and failure data.

On the other hand, determining execution time of hard real-time system has been thoroughly investigated during the past two decades by the real-time systems community (Alan C. Shaw 2001, Puschner et al. 2000). In order to guarantee the satisfaction of the deadlines of such systems, designers need to analyze/measure the execution time of the different tasks of the system. This can be done using static code analysis or measurement during execution prior to deployment. Worst Case Execution Time (WCET) analysis determines the upper bounds of the execution time of the different tasks (Puschner et al. 2000). This is usually done by characterizing the different paths in the program and the constituent instructions (machine level) and the execution time of each instruction. This analysis is generally hardware dependent. The analysis of the execution time of upgrade campaign specification is different in a sense that it is at a higher level and takes into account failures and retrials. In addition, we also determine the service outage in our simulation.

7 CONCLUSION

Availability is an important requirement for carrier grade services. Upgrade campaigns can cause outage and therefore require evaluation beforehand. Upgrade campaigns that will jeopardize the required availability will be eliminated in the selection/elimination process. An evaluation based on random simulation lacks accuracy, thus there is a need for some tie breaking scenarios. The best case scenario is a valid choice, useful for eliminating upgrade campaign specifications, it is simpler to simulate and takes less time, but remains inconclusive in a fair amount of cases. The use of the worst case scenario is more effective and gives useful insight on the applicability of the campaign at both execution time and service outage levels as well as opportunities to further optimize an upgrade campaign specification. Our approach provides a system administrator with a set of upgrade campaign specifications applicable with respect to the acceptable outage and the maintenance window. The final word is with the administrator as he/she can favor one criterion over the other.

ACKNOWLEDGMENT

This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson.

REFERENCES

Ajmone Marsan, M., Chiola, G. 1987. *On Petri Nets with Deterministic and Exponentially Distributed Firing Times*. In: Rozenberg, G. (ed.) APN 1987. LNCS, vol. 266, pp. 132–145. Springer, Heidelberg.

Alan C. Shaw. 2001. *Real-time Systems and Software*. Wiley.

Ali Kanso, Maria Toeroe, Ferhat Khendek. 2012. *Configuration-Based Service Availability Analysis for Middleware Managed Applications*. SAM, Volume 7744 of the series Lecture Notes in Computer Science pp 229-248.

Ali Kanso, Maria Toeroe, Ferhat Khendek. 2013. *Automating Service Availability Analysis: An Application to a Highly Available Media-Streaming Service*. SERE-C, IEEE 7th International Conference on, pp. 94–101.

Bernard P. Zeigler, Herbert Praehofer, Tag Gon Kim. 2000. *Theory of Modeling and Simulation, Second Edition*. Academic Press; 2 edition.

Cisco Forum. https://supportforums.cisco.com/discussion/12483781/time-estimate-pcd-upgrade-cucm (Posted April 20, 2015)

DEVS-Suite. http://acims.asu.edu/software/devs-suite/

Louis M. Rose, Richard F. Palge, Dimitrios S. Kolovos, Fiona A. Polack. 2008. *The Epsilon Generation Language*. ECMDA-FA '08 Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications.

Maria Toeroe, Francis Tam. 2012. *Service Availability: Principles and Practice*. Wiley.

Michal Fornadel, Peter Lacko, Andrej Danko. 2013. *Estimation of Legacy Application Upgrade Time using Evolutionary Approach*. Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on.

Puschner, Peter, and Alan Burns. 2000. Guest editorial: *A review of worst-case execution-time analysis*. Real-Time Systems 18.2: 115-128.

Service Availability Forum, SAF 1. http://www.saforum.org

Service Availability Forum, SAF 2. 2011. *Availability Management Framework specification*. SAI-AIS-AMF-B.04.01.AL.

Service Availability Forum, SAF 3. 2011. *Software Management Framework specification*. SAI-AIS-SMF-A.01.02.AL.

Sungung Kim, Hessam S. Sarjoughian, Vignesh Elamvazhuthi. 2009. *DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring*. SpringSim '09 Proceedings of the 2009 Spring Simulation Multiconference.

Xiangqun Qiu, R. Telikepalli, T. Drwiega, J. Yan. 2005. *Reliability and Availability Assessment of Storage Area Network Extension Solutions*. IEEE Communications Magazine , Volume:43 , Issue: 3, pp. 80-85

AUTHOR BIOGRAPHIES

OUSSAMA JEBBAR received his Engineering diploma from the National Institute for Telecommunication (INPT, Rabat, Morocco) in September 2014. He completed his M.A.Sc. degree in software engineering at Concordia University in December 2016. He worked on SAF compliant systems and their upgrade, He is now pursuing a PhD degree in the same institution.

FERHAT KHENDEK received a PhD degree from Université de Montréal in Canada. He is a full professor in the Department of Electrical and Computer Engineering at Concordia University where he also holds since 2011 the NSERC/Ericsson Senior Industrial Research Chair in Model Based Management, a major collaboration between Ericsson and Concordia University. Ferhat Khendek‘s research interests are in model based software engineering and management, formal methods, validation and testing, cloud computing, real-time software systems, and service engineering and architectures.

MARIA TOEROE is an Expert at Ericsson working in the area of dependable software, service availability and fault tolerance. She has represented Ericsson in the Service Availability Forum and more recently in OPNFV. Maria is also the technical coordinator of the Ericsson research collaboration with Concordia University. She has numerous publications and served as organizer and program committee member of different conferences. Maria holds a PhD from the Budapest University of Technology and Economics.