

Availability Management Framework

DESCRIPTION

Copyright

© Ericsson AB 2018–2019. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

Trademark List

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



Contents

1	Understanding Availability Management Framework	1
1.1	Basic Availability Management Framework Concepts	1
1.2	Application	1
1.3	Cluster and Node	2
1.4	Component and Service Unit	2
1.5	Health Monitoring	2
1.6	Workload	5
1.7	Assignment	5
1.8	Failover and Switchover	6
1.9	Error Detection, Recovery, Repair, and Escalation	6
1.10	Network Partitioning	7
1.11	Information Model	9
1.12	Redundancy Model	9
1.13	Administrative Operations	10





1 Understanding Availability Management Framework

1.1 Basic Availability Management Framework Concepts

The Availability Management Framework (AMF) is about managing applications and to keep the service they provide available always. This includes the following major responsibilities:

- Control the life cycle: start and stop the application.
- Monitor the health of started applications.
- Manage the workload.
- Recover and repair of failed services.
- Support administrative operations on modeled entities.
- Send alarms.
- Manage the system model for applications.

1.2 Application

By application in the AMF context is usually meant the server part in a client-server application. There are many types of servers such as web servers, database servers, and gaming servers.

Green field applications are applications written from scratch possibly with the AMF integration in mind. If so, they can freely use the AMF concepts depending on their ambition level to provide service availability and become Service Availability-aware (SA-aware).

Third-Party Programs (3PPs) or legacy applications are applications that exist and that are not integrated with the AMF. Such applications are referred to as non-SA-aware. Certain features exist in the AMF to support integration of these types of applications. Such integration is important to provide a complete highly available system solution that includes databases and storage solutions.

The AMF environment is a clustered environment and the application can be distributed in a cluster. The subparts of a distributed application do either or both of the following:

- Share resources such as a file system or a database.
- Communicate with each other to provide the high-level service.



An AMF application can consist of only a single operating system process but this gives quite a bit of overhead because of the AMF modeling requirements. It is, however, a good starting point when there are plans to make the application High Availability (HA) or distributed, or both.

1.3 Cluster and Node

Cluster and Node are logical entities of the AMF system model. An AMF node corresponds to an operating system instance. The set of AMF nodes form the AMF cluster. Nodes in a cluster belong to the same communication subnet; no routing is needed within a cluster.

1.4 Component and Service Unit

The component is one logical entity of the AMF system model. A component represents a program in execution under control of the AMF. Usually a component corresponds 1:1 to an operating system process.

The term SA-aware component is used to describe a component that is integrated with and using the API.

Components are grouped into Service Units (SUs), a logical entity completely associated with an AMF node. All components in an SU execute on the same AMF node.

1.5 Health Monitoring

Health monitoring is important to achieve service availability and is used to detect errors and anomalies in the system. Monitoring is always performed on a per component basis and is also called component monitoring.

The AMF supports three different types of monitoring:

- Passive
- External active
- Internal active

With active monitoring, latent faults, such as a looping and not responding program, can be detected, which is not the case using passive monitoring.

When active monitoring is used, it is also possible to validate the data received from the service monitored. For example, if system uptime is requested from an SNMP agent (because of active monitoring of it), the result can be validated and checked to see if it is reasonable. This kind of monitoring is out of the scope of the AMF and this document, besides it is service-specific. If used, it gives even higher service availability because another class of errors can be detected.



The recovery action taken by the AMF when a fault has been detected is configurable but can, for example, be `COMPONENT_RESTART`. If a monitored process dies, it is restarted again by the AMF. A recommended recovery action can also be specified in the API used to report errors.

For more information, refer to **AIS AMF Specification** at <http://www.saforum.org>.

1.5.1 Passive Monitoring

In passive monitoring, the AMF uses operating system features to assess the health of a component. Currently only monitoring the death of a process is defined but one can envision monitoring other system resources like main memory use.

As operating system features are used, the component is not actively involved in the monitoring and its code is not instrumented, hence the name passive monitoring.

The AMF implicitly performs passive monitoring on SA-aware components. If such a component dies, for example because of segmentation fault, the AMF automatically detects it.

To use passive monitoring for other types of components (or for a subprocess), it must be started using function `saAmfPmStart()` and stopped using function `saAmfPmStop()`.

The time between fault and detection is implementation-specific and cannot be configured using either the API or through configuration objects.

1.5.2 External Active Monitoring

In external active monitoring the component code is not instrumented, instead an external entity called a monitor is used to assess the health of the component.

The monitor is preferably sending real service requests to the monitored component and supervising that a correct response is received in a timely manner.

An AMF component can be configured with optional Application Monitoring (AM) commands. Command `AM_START` is called after the `initiate` command and `AM_STOP` is called before the `terminate` command.

`AM_START` starts a monitor process that periodically assesses the health of the monitored application by making a simple service request to it. The AMF is not involved in the actual monitoring, that is, the responsibility of the monitor process.

When the monitor detects a health problem with its monitored service, it is to call function `saAmfComponentErrorReport()`. This implies that the monitor itself is written in C/C++ or that a helper command exists that wraps `saAmfComponentErrorReport()` so that it can be called by a script implemented monitor.

In this case no one monitors the monitor, but as the monitor is simple and small it can probably be considered fault free by review. If this is not appropriate, the monitor can be implemented as an AMF SA-aware component to which the AM commands send monitoring requests.

For more information, refer to **AIS AMF Specification** at <http://www.saforum.org>.

1.5.3 Internal Active Monitoring

Using internal active monitoring, the component must be specifically designed. The purpose of such code is to monitor the component health and discover latent faults. The execution of such code (often called audits) is in the AMF called a health check.

As the code is instrumented, this type of monitoring is normally only used for SA-aware components.

A health check can be triggered by the component itself or by the AMF. When triggered by the AMF, health check requests are sent periodically to the component with a certain configurable period. The AMF expects a response within a certain configurable time called the duration. The duration is always shorter than the period.

A component can have several health checks active at the same time. Each health check is identified by a key – a name. Some reasoning for this: depending on the check performed, the impact on the service provided varies. A normal service request has little impact and can be run with a shorter period. More detailed component audits can have more service impact and are to be run with a longer period.

Active monitoring means that the provided service is to be checked. Therefore, health checks cannot be acted on by, for example, a separate decoupled thread in the component, unless it actually does a service request internally.

Configuration of period (and duration) must be done with high load in mind. It is a trade-off between fast true error detection and avoidance of false error detection. A longer period is good to avoid false error detection but it takes longer to detect latent faults. A health check period is normally in the second range or even 10 s of second range, it is most likely not less than a second. The health check duration most likely must be longer than the callback time-out, typically twice as long. It depends on the AMF implementation if two supervision timers run at the same time or if health checks are skipped when some other supervision is active, for example, callback time-out.

An unexpected death of the registered process for an AMF component is instantly detected by the AMF and requires no active monitoring.

Errors are reported to the AMF in two ways. When the AMF invoked health checks are used, a negative response is given using function `saAmfResponse()`. When component invoked health checks are used, the component responds with a negative response using function `saAmfHealthcheckConfirm()`.



For more information, refer to **AIS AMF Specification** at <http://www.saforum.org>.

1.6 Workload

A normal non-AMF-aware program provides service directly when started. There is no distinction between the program and the service it provides. However, if the service or work the program performs can be categorized and quantified, it can also be modeled and managed. This categorized and quantified work/service is what the AMF means by workload. Workload is a core concept used by the AMF to enable high availability and is important to understand. When an application uses the workload concept, the AMF enables for sophisticated redundancy schemes.

An application designed with the workload separation in mind is called SA-aware in AMF terms. That is, it can be started and be “idle” – do nothing until the AMF tells it to be active or standby for a certain workload.

A simple example can be a web server that starts and initializes but does not bind to port 80 until assigned the corresponding active workload. On another node, the same program can be running as standby waiting to be activated if the other instance goes down. This is an example of a simple 2N redundancy scheme.

With AMF concepts, the workload is called a Service Instance (SI) and these are assigned to SUs. An SI is further broken down in to Component Service Instances (CSIs), which are assigned to components (processes) and visible in the API for the program designer.

1.7 Assignment

The AMF assigns a workload in active or standby state to an application. This means that the application upon receiving the assignment is to start providing service according to the state of the assignment, and the amount and type of service as described by the workload.

For simplicity, the application is often designed so that, when assigned an active workload, it already knows the amount and type of service the workload represents. In the web server example, the active workload means bind to port 80. But if the bind port number can vary, the AMF concepts for categorizing the workload can be used.

More complex schemes can be used to describe workloads. For example, a workload can describe a range of subscribers and their certain properties. Then one can imagine some application “workers” collectively providing the high-level service, each worker contributing with its little piece and all workers together provides the complete service.

1.8 Failover and Switchover

In this section, the “operator” can mean either a human or a management application running within the system such as software management.

Failover means an unexpected reassignment (from an operator point of view) of a workload to another instance of an application. In AMF terms, the SI is reassigned to another SU. Failover is always a consequence of a fault in the system of which the AMF is aware.

Switchover means an expected reassignment of a workload to another instance of an application. It is expected because it is either initiated by an operator or by the AMF itself. When recovering from a fault, the AMF can fail over some SIs and switch over others. This occurs in some conditions, always as a consequence of a fault and depending on the application model and configuration. This is to minimize disturbance in the system.

A switchover is supposed to be less intrusive to the service provided by the application. SA-aware components are to be designed with this objective.

1.9 Error Detection, Recovery, Repair, and Escalation

Error detection is the responsibility for all entities in the system.

After an error has been detected and reported, the AMF tries to recover the application provided service from the error. Recovery is performed automatically by the AMF to ensure that all assignments are reassigned to a non-erroneous component. If the AMF cannot reassign the workload, it sends the alarm “workload unassigned”, which means that a service is not available at all.

A recovery action can be recommended when an error is reported. A default action is also configured for the component. The executed action is never weaker than the one recommended but can be stronger.

Normally the first level of recovery is restart of the erroneous component. The objective is to avoid reassigning the workload to another component. If component restart fails or another error occurs within the component probation time, the next action – because of escalation – is restart of the whole SU.

If the SU is restarted too many times during the SU probation time, the recovery action is escalated to failover.

If restart is disabled by configuration or the restart failed, the next level of recovery action – because of escalation or recommendation – is failover. This means assigning the workload to another SU than the failed component belongs to. The failover scope can because of escalation be extended from SU to node (all SUs hosted by a node).

After recovery, repair is by default automatically performed on the erroneous entity. By configuration, automatic repair can be disabled and thus make the



responsibility a non-AMF issue. Restart recovery actions are considered as repair actions and no further action is needed. However, if the recovery action was failover, the AMF tries to reinstantiate the component and possibly reassign it.

For more information, refer to **AIS AMF Specification** at <http://www.saforum.org>.

1.10 Network Partitioning

1.10.1 Consistency over availability

Core MW prioritizes consistency over availability. If a node becomes unreachable to the active SC due to network partitioning, if possible, it will be rebooted when the node becomes reachable. The rationale is that during the partitioning, the node would have lost ability to keep up with state and other data changes occurring in the main network partition.

1.10.2 Relation to RL2 Feature

Network partitioning is less of a problem when the RL2 feature is disabled. If a payload loses connectivity to all SCs, it will self-fence by rebooting. However, if a payload with RL2 feature disabled can still see the standby SC, split-brain is still an issue.

If the RL2 feature is enabled, an isolated PL will remain running and can pose a problem in terms of data consistency due to split-brain.

Please see Node Failover Timer for one possible mitigation strategy. When an isolated PL is merged back with the main network partition, it will be rebooted by the AMF director on the active SC.

1.10.3 Remote Fencing

Remote fencing can be used to avoid SC split-brain when there is network partitioning. When the active SC notices that connectivity to the standby SC is lost, it can remotely fence the standby SC to help avoid split-brain.

When remote fencing is used in conjunction with Node Failover Timer feature, payloads can also be remotely fenced by the active SC.

1.10.3.1 TIPC Link Loss timer

The TIPC Link Loss timer as the name suggests is a timer which specifies the amount of isolated time allowed on TIPC link before it is decided that the link is down. For more information refer to LDE documentation.



1.10.3.2 Promote Active Timer

This timer controls the period a standby SC will wait before it promotes itself to be the active SC.

If the active SC reappears before the end of this period, then the standby SC will self-fence by rebooting itself to order to maintain data consistency.

This parameter can be used to aid prevention of SC split-brain. That is, the presence of multiple active SCs in the cluster.

If the network disturbance between the active SC and standby SC is temporary and the duration is smaller than the Promote Active Timer, then SC split-brain is prevented. However, if the network disturbance is longer, then SC split-brain will still occur.

The promote active timer will affect the response time of the standby SC in taking over the active role in a real active SC failure.

Thus, it is a balancing act to keep this timer large enough for expected network disturbance durations but keep it small enough to have a reasonable response time to actual active SC failures.

The main benefit is to postpone the promotion of stand-by SC to become active SC for a desired time specific to system and demand and to delay the failover of assignment.

For more information Refer to Section 2.2.18 Network Partitioning in OpenSAF Availability Management Framework.

For more information on how to configure this timer, refer to Section 3.11.2 in Core MW Management Guide.

1.10.3.3 Node Failover Timer

Refer to Section 3.3 (OsafAmfDelayNodeFailoverTimeout) in OpenSAF Availability Management Framework.

For more information on how to configure this timer, refer to Section 3.11.2 in Core MW Management Guide.

1.10.3.4 Node Wait Timer

Refer to Section 3.3 (OsafAmfDelayNodeFailoverWaitTimeout) in OpenSAF Availability Management Framework.

For more information on how to configure this timer, refer to Section 3.11.2 in Core MW Management Guide.



1.11 Information Model

An SA-Forum system is managed through an information model. The information model consists of managed objects that represent various logical entities in the system.

The information model is managed by the Information Model Management service (IMM). It is out of scope of this document to describe the IMM. For more information, refer to **AIS IMM Specification** at <http://www.saforum.org>.

Most SA-Forum specified services defines an information model. This is particularly true for the AMF that defines a rich information model to support application modeling. For more information, refer to **AIS AMF Specification** at <http://www.saforum.org>.

The IMM supports administrative operations, which can be seen as a Remote Procedure Call on an object in the model. An operator, for example, stops an application because it is about to be upgraded.

An application can also use the IMM to store its specific configuration data, thus making it possible to configure and manage in SA-Forum intended way.

1.12 Redundancy Model

The AMF provides the concept of redundancy models. The redundancy model helps the AMF to keep the application service available per its requirements.

Historically telecommunications applications have been designed to have standby entities. The AMF support those types of applications by providing redundancy models that include standby workload assignments. Other SA-Forum services, such as the Checkpoint service (CKPT), provide means to make standby entities “warmer” – more ready to take over an active assignment. The CKPT enables an application to replicate its state data.

By leveraging on the separation of program and workload, the AMF can manage many instances of a program and transfer the active workload from a non-operational program to an operational program.

The following redundancy models are defined:

- 2N
- N+M
- N-way
- N-way active
- NoRedundancy

For more information, refer to **AIS AMF Specification** at <http://www.saforum.org>.



1.13 Administrative Operations

The AMF model specifies quite a few administrative operations defined for certain entities. The AMF is the implementer of such a call with help and cooperation from the affected application component or components.

Examples of administrative operations are LOCK and UNLOCK for workload management, but other operations also exist.

Administrative operations are needed so that an operator can communicate and control the AMF. For example, upgrading an AMF application without involving the AMF causes the AMF to consider the application to have failed.

Administrative operations are used by an operator or more likely a management program acting on behalf of an operator at a network management system.

One example of the latter is software management. When a program is upgraded, it is locked, updated, and finally unlocked again.

For more information, refer to **AIS AMF Specification** at <http://www.saforum.org>.