Implementation of the UniFAFF framework for Autonomic Fault-Management in ANA Networks

Ranganai Chaparadza, Nikolay Tcholtchev, Ina Schieferdecker
MOTION
Fraunhofer FOKUS Institute for Open Communication Systems
Berlin, Germany
{Ranganai.Chaparadza; Nikolay.Tcholtchev;Ina.Schieferdecker}@fokus.fraunhofer.de

Abstract— **Research in the area of Self-Managing Networks is on the rise. Some research initiatives are calling for clean-slate/revolutionary designs of future networks (e.g. the future internet) while some initiatives are calling for an evolutionary approach and are focused on making today's networks evolve by introducing new communication paradigms into the network models and architectures in an incremental way. For both types of initiatives towards the design of future networks, one of the most important questions being raised is: "How can the well known and very successful FCAPS network management framework be adopted and extended as necessitated by emerging network architecture designs for Self-Managing Networks?". Recently, a framework called UniFAFF has emerged, that answers the posed question with respect to moving from today's Fault-Management whose processes are not autonomic, to making Fault-Management processes become autonomic as necessitated by Self-Managing Networks. UniFAFF stands for Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks. This paper reports on the first attempt to implement the UniFAFF framework for a clean-slate type of network design— namely the ANA network architectural design.**

Keywords — self-manging networks, Autonomic Network Architetures, Autonomic Fault-Management and Failure-Detection

I. INTRODUCTION

The networking community is seeing a rise in research in the area of Self-Managing Networks. Some research initiatives are calling for clean-slate/revolutionary designs of future networks (e.g. the future internet) while some initiatives are calling for an evolutionary approach to future-internet design and are focused on making today's networks evolve by introducing new communication paradigms into the network models and architectures in an incremental way. For both types of initiatives towards the design of future networks, one of the most important questions being raised is: "*How can the well known and very successful FCAPS network management framework [1] be adopted and extended as necessitated by emerging network architecture designs for Self-Managing Networks?*". Recently, a framework called UniFAFF [2] has emerged, that answers the posed question with respect to moving from today's Fault-Management whose processes are not autonomic, to making Fault-Management processes become autonomic as necessitated by Self-Managing Networks. UniFAFF stands for Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks. UniFAFF defines a set of criteria upon which the framework is founded and presents a generic architecture created out of unifying related work and understanding in the field of Fault-Management and Failure-Detection. The UniFAFF calls for the design and evaluation of the concepts, components and interfaces of its generic architecture. The UniFAFF framework provides the following understanding and definition: We talk about *Autonomic Fault-management* and *Failure-Detection* when there are cooperative and collaborative mechanisms implemented in nodes and the network as a whole, to automatically detect faults, errors and failures, share knowledge about such incidents, diagnose or localize faults, as well as remove faults, throughout the operation lifetime of a node and the network. A *M.Sc. thesis* [3] was carried out as the first attempt to implement the UniFAFF framework. In [3], some of the key components of UniFAFF have been designed and evaluated, and the algorithms proposed for incident information and alarms dissemination to enable information/knowledge sharing among network entities, have also been designed and evaluated - with a focus on flooding and gossip algorithms. The components and mechanisms developed in the thesis [3] were evaluated on ANA networks [5]; a kind of networks which are based on clean-slate type of network designs, though UniFAFF can also be applied to today's evolving networks. This paper briefly describes the approach that has been taken in implementing UniFAFF for ANA networks [4]. Due to limited space, we point out only the key issues and point the reader to [2][3][4] for more detailed information.

This paper is organized as follows: In **Section II** presents ANA Networks in brief. **Section III** presents the UniFAFF framework in brief. **Section IV** presents the Implementation of Selected Key Components of the UniFAFF framework for ANA Networks. **Section V** presents some Example Scenarios for Autonomic Fault-Management. **Section VI** presents the Evaluation of Selected Key Components of the UniFAFF framework Implemented for ANA Networks. **Section VII** gives some Conclusions and further work in implementing the UniFAFF framework for clean-slate type of network designs.

II. ANA NETWORKS IN BRIEF

Autonomics in ANA, a clean-slate architecture, is mainly introduced through a set of concepts that should enable the

self-management of a system by allowing for dynamic composition of node behaviors and network behaviors via the use of functional composition frameworks intrinsically meant to operate in the network nodes. One of the key concepts of abstractions introduced in ANA is the concept of a *Functional Block (FB)*, which is basically a functional entity that by design has the ability to generate, consume, process or forward information received on one or more of its input channels. An atomic Functional Block, as opposed to a composite Functional Block, is called a *Brick*. So-called *Functional composition framework(s)* [6] governing the composition of the protocol stacks and the overall behaviors of the nodes, facilitates the ability for a node to exercise reloading faulty entities and (re)-composing different functional entities and behaviors of the node as necessitated by challenges in the network operation as well as context changes. One of the most notable and revolutionary ideas of ANA is the concept of *flexible protocol stacks*. The other concept introduced in ANA is the concept of a *Compartment*, which is defined as a policed set of *Functional Blocks (FBs)*, *Information Dispatch Points (IDPs)* and *Information Channels (ICs)*, which enables communication for its members according to some commonly agreed set of communication principles, protocols and policies. It can also be thought of as a "*realm*".

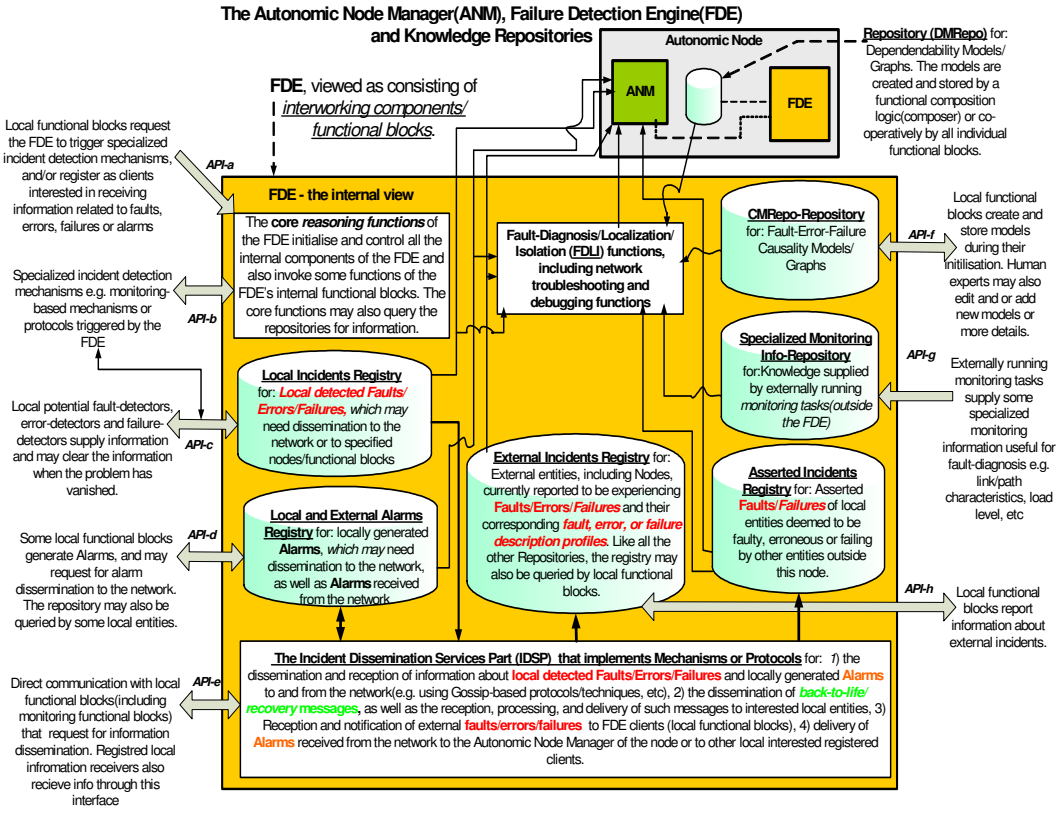

Figure 1: The Generic Architecture of the UniFAFF framework for an Autonomic Node

III. THE UNIFAFF FRAMEWORK IN BRIEF

UniFAFF stands for *Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks* and comprises a set of components that aim to facilitate Autonomic Fault-Management while abstracting from the underlying network type. UniFAFF [2] proposes to call this set of components the *Failure Detection Engine (FDE)*. The key principle behind the design of the UniFAFF is as follows: *"In a self-managing node or network, Failure-Detection and Fault-Management must be co-operatively and/or collaboratively handled by a number of functional entities through the sharing of "**knowledge**" or **"information"** about failures, errors, faults, their points of occurrence or manifestations, their causality relationships, dependency relationships between entities (protocols, services, nodes, etc.) and context information etc. Functional entities have to use all such knowledge in order to execute or co-operatively request each other to execute some **assigned fault-management related functions**. In an autonomic node or network, there is a need for functional entities to co-operate and/or collaborate in Fault-Detection, Error-Detection, Failure-Detection and Fault-Localization since one entity may be able to detect an incident within a time frame shorter that other entities can detect it (if at all)."*

Therefore, UniFAFF [2] addresses the following issues:

(A) The *processes* involved in Autonomic Fault-Management, namely: (1) *Automated Alarm-Generation*; (2) *Automated Incident-Detection*; (3) *Automated Alarm/Fault/ Error/Failure Dissemination*; (4) *Automated and Collaborative Fault-Diagnosis /Localization/Isolation*; (5) *Automated Fault-Removal*.

(B) The definition of a set of *extensible criteria* upon which the UniFAFF framework is founded.

(C) The specification of "*Requirements*" that must be followed by designers of functional entities of an autonomic node, such as the designers of Functional Blocks for ANA nodes.

(D) The Meta-Models i.e. Information Models that describe the kind of information/knowledge which needs to be generated and shared among reactive entities of the autonomic nodes/networks that act upon the information/knowledge. [3] presents the design of the Meta-Models (i.e. Information models) required by the UniFAFF framework.

(C) The provision of a *Generic Architecture* (see **Figure 1**) for implementing autonomic fault-management and failure-

detection for self-managing networks. Note that the architecture relates to a single autonomic/autonomous node.

For the ANA architecture, the FDE part of the UniFAFF framework was further split into two parts namely: the Challenge Detection Engine (CDE) that actually comprises of all the repositories and the FDLI functions of the FDE, and the Incident Information Dissemination Engine (IDE) part of the FDE (i.e. the IDSP). [3] also presents some flooding and gossip algorithms that were developed for the IDE. For more information on the specifications of the API functions, as well as mechanisms employed by the FDLI functions for distributed fault-diagnosis we refer the reader to [3]. **Figure 2** gives an overview of the relations among the components implemented for the ANA architecture, including the FDLI functions of the FDE.

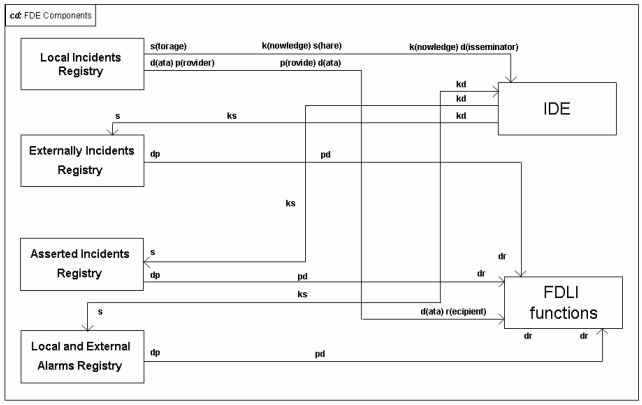

Figure 2: The relationships among the basic components of the FDE

In order to illustrate the relationships between the fragments on a lower level, the CDE boundaries have been left out of this component view. The following roles and relations have been defined among the objects in the diagram to aid understanding:

- **Relations**
  - **Knowledge Share (ks)** – this association merely defines that two components are sharing knowledge in a bidirectional or a unidirectional way depending on the arrow used to connect them.
  - **Provide Data (pd)** – this relation specifies that one object provides information to another.
- **Roles**
  - **Storage (s)** – this role is dedicated for the repositories in order to mark them as the instance(s) responsible for storing the information.
  - **Knowledge Disseminator (kd)** – this is the task of the IDE in the system and establishes it as the component responsible for information dissemination.
  - **Data Provider (dp)** – the role of the repositories with respect to the FDLI block, meaning that each repository has to provide knowledge concerning incidents and alarms and respectively notify the FDLI functions, so that they can start a process of Fault-Isolation upon the arrival of new alarm/incident information.
  - **Data Recipient (dr)** – the FDLI functions act as a data recipient, "waiting" (the FDLI is actually implemented as a library) to get new incident information from the repositories in order to start a Fault-Diagnosis/Localization/Isolation. This machinery is realized as a callback mechanism.

The associations/roles in **Figure 2** describe the relations between the components and are meant to illustrate the general fact that each system part can either passively provide knowledge/data when being requested/queried or can actively disseminate newly arrived data to the interested entities using a kind of *listener/callback* mechanism.

IV. IMPLEMENTATION OF SELECTED KEY COMPONENTS OF THE UNIFAFF FRAMEWORK FOR ANA NETWORKS

We considered the components for the storage of alarms, faults, errors, and failures as well as the engine for their dissemination as the vital basics for Autonomic Fault-Management. Therefore, the main focus was set on the implementation of the *Local Incidents Registry*, the *External Incidents Registry*, the *Asserted Incidents Registry*, the *Incident Information Dissemination Engine* part of the FDE, as well as on creating the hooks and the interfaces to the *Fault-Diagnosis/Localization/Isolation functions* and the *Autonomic Node Manager (ANM)* part of an autonomic node that composes, manages and oversees the overall behavior of the node's functional entities. The figures that follow present some selected samples of information flow and interaction flow sequence diagrams that illustrate the functions of the API's that were specified and designed for the ANA architecture. On the diagrams, the operation/primitive "*reasonForAnIncident*" or "*reasonForAnAlarm*" is meant to request the FDLI functions to reason about an incident or alarm i.e. find out the cause.

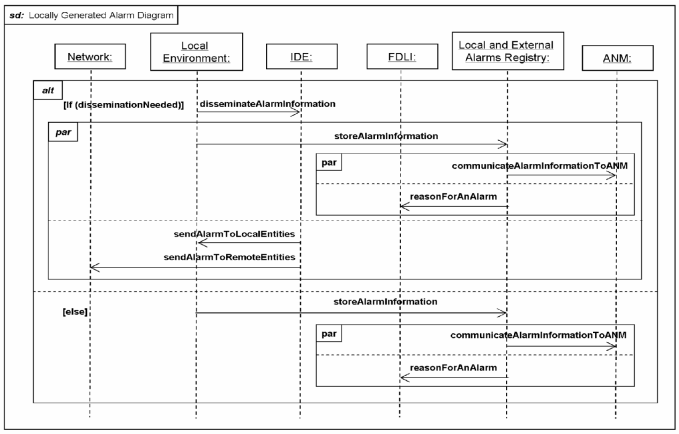

Figure 3: Sequence diagram for *Alarm* handling within the FDE after a local functional block has generated an alarm.

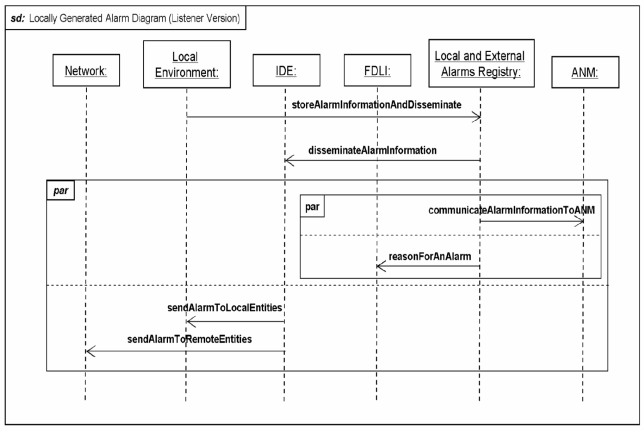

Figure 4: Sequence diagram of the listener version for alarm handling within the FDE after a local functional block has generated an *Alarm*

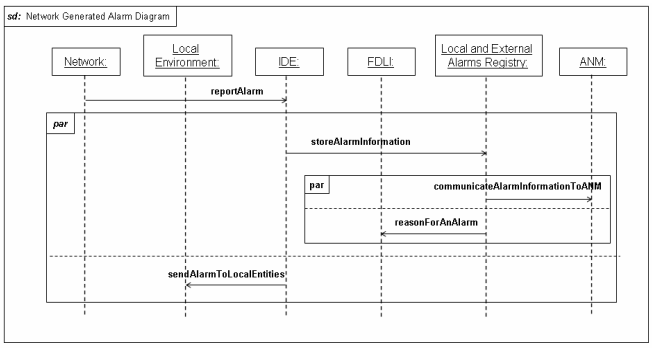

Figure 5: Processing a remotely generated alarm

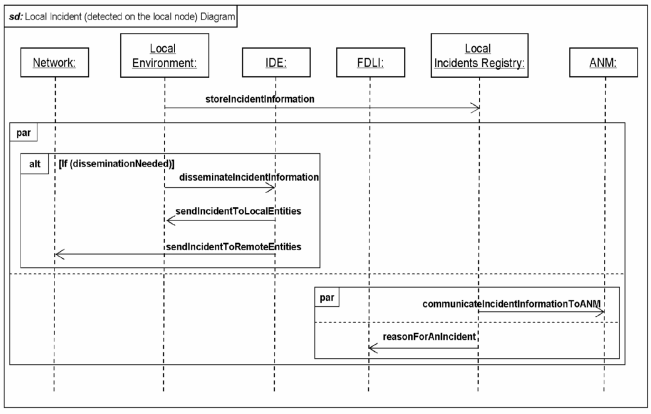

Figure 6: Processing a local incident that was detected on the local node

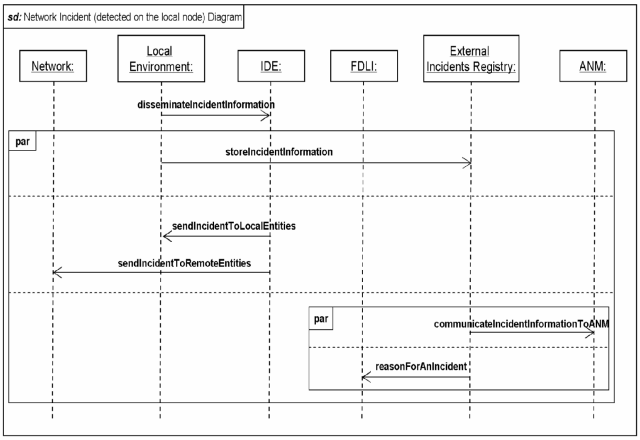

Figure 7: Handling of a network incident that was detected on the local node

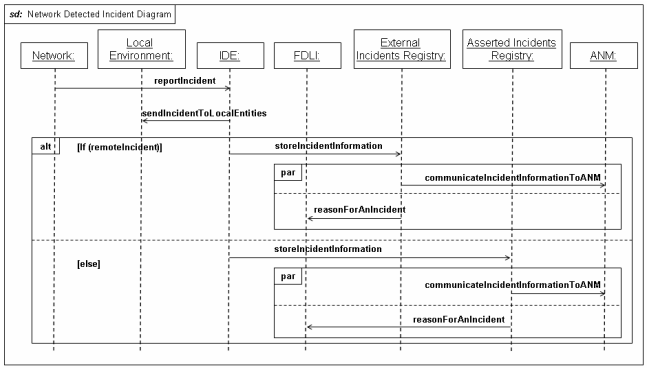

Figure 8: Processing of incident information coming from the network via the local IDE

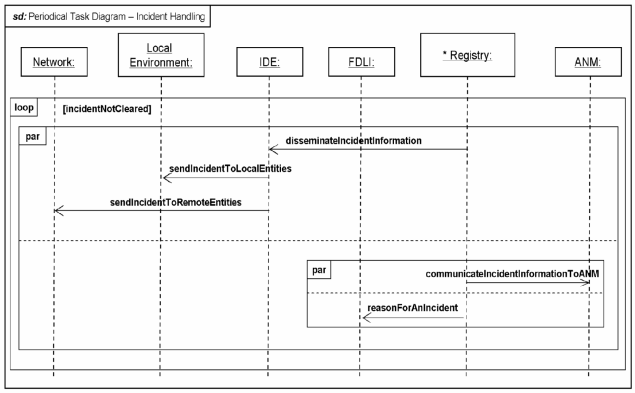

Figure 9: The actions to be undertaken by a periodical behaviour of an Incident storage Repository concerning any uncleared incidents. * denotes an arbitrary registry and the flag "incidentNotCleared" contains the state of the particular incident – failure/error/fault.

V. EXAMPLE SCENARIOS FOR AUTONOMIC FAULT-MANAGEMENT

The following figures (figures 10 and 11) illustrate simple cases of a complete sequence of actions on how autonomic Fault-management can be realized, from the processes of incident detection through to the ultimate goal of fault-removal, of which in the cases presented, is simply achieved through

reloading of a faulty entity after it has been localized by FDLI functions.

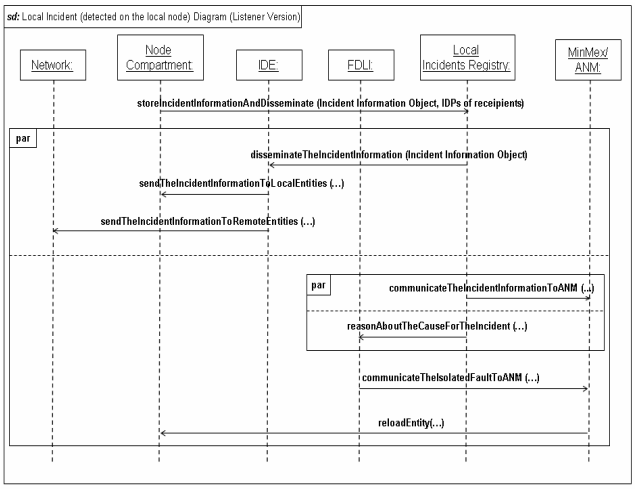

Figure 10: Local incident scenario - An abstract scenario of *Autonomic Fault-Management* in the context of the UniFAFF framework.

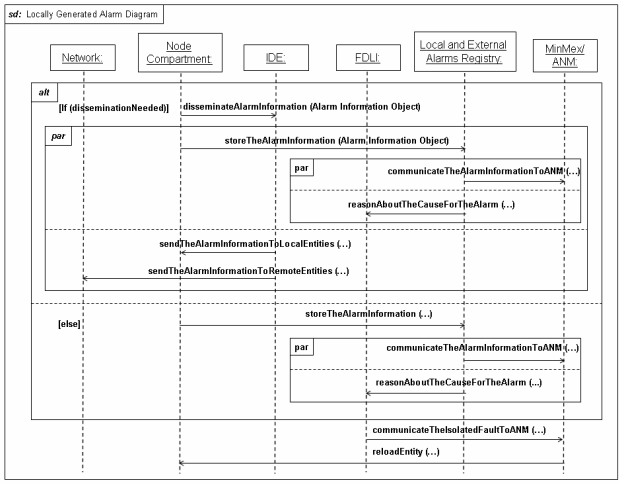

Figure 11: Locally generated alarm scenario - An abstract scenario of *Autonomic Fault-Managemen*t in the context of the UniFAFF framework.

VI. EVALUATION OF SELECTED KEY COMPONENTS OF THE UNIFAFF FRAMEWORK IMPLEMENTED FOR ANA NETWORKS

Figure 12 provides a description of the test scenario used to validate the functionalities of the implemented FDE bricks. The simulation facilities of the ANA project (vlink) were used to create the test scenario. A vlink virtual physical layer was created on a single computer (Intel(R) Pentium(R) M processor 1.60GH, 509.2 MB RAM), and three ANA nodes were attached to it. The ANA nodes were then able to communicate over the vlink using the *Ethernet compartment*. Four client bricks were created for evaluation purposes.

- **usrBrickFDE1** – This brick acts as an incident-detector and alarm-generator and submits the corresponding information to the CDE repositories on the local node. Subsequently this brick queries the repositories to check whether the retrieval of knowledge works correctly. At the end, the brick submits again incident and alarm knowledge to the registries thereby using the listener mechanisms of a repository to trigger the dissemination of the knowledge to interested local clients and to the network.
- **usrBrickFDE2** – This brick registers for receiving information of interest from the Local Incidents Repository.
- **usrBrickFDE3** – This brick registers at the IDE of the local node (using API-e) for receiving information about incidents and alarms stored in the repositories.
- **usrBrickFDE4** – This brick registers at the External Incidents Repository for receiving information of interest upon its arrival.

The order of the actions that are issued by the different test bricks is given by their enumeration in **Figure 12**. Next we go through the numbered processes triggered by those actions:

1 stands for "Registering at the Local Incidents Repository in order to receive Incident Information & Registering at the Alarms Repository for receiving Alarm Information".

2 stands for "Registering at the IDE for receiving Alarm/Incident Information".

3 represents the process of "Registering at the External Incidents Repository & Alarms Repository for receiving Incident & Alarm Information of Interest".

4 stands for "Storing Alarm & Incidents Information into the FDE Repositories".

4.1 represents the process of "Querying the FDE Repositories".

5 stands for "Storing Information in one of the Incident Repositories and Triggering Dissemination".

5.* denotes all the listener/callback information consumer actions that follow **5** and could be named "Getting Incident/Alarm Information from the corresponding Repository after a successful registration".

The bricks and the test environment described in **Figure 12** have been used to execute different performance test and to check the stability of the UniFAFF components, designed and implemented in [3]. In order to achieve this, the brick acting as an incident-detector and an alarm-generator has been adjusted, so that it behaves as if "detecting" an incident and generates an alarm every three seconds and repeats this procedure 255 times. Every time the submitted information ran through the whole chain – **1**, **2**, **3**, **4**, **4.1**, **5**, and **5.*** - thereby examining the stability of all possible aspects like listener mechanisms, querying, dissemination of information to interested local clients, dissemination of information over the *Ethernet compartment* using flooding or gossiping, etc. Thereby the system proved to be stable in general. However, some small problems were observed:

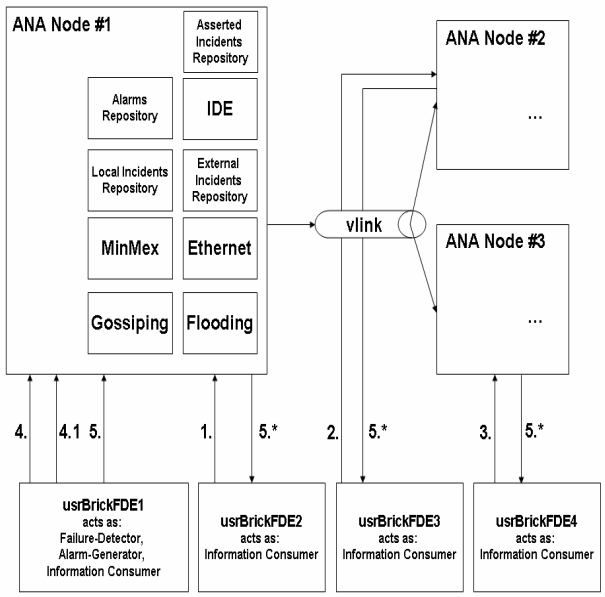

Figure 12: A description of the test scenario used to validate the functionalities of the FDE

- The periodical tasks (behaviour) of Repositorries/Registries (**see Figure 9**) that are responsible for the dissemination of uncleared alarm/incident information managed to overload the IDE with the dissemination of more than 200 event models at one point in time. This means that some mechanism for restricting the amount of information sent at once by the periodical task of a repository has to be considered (a subject for further research).

- The other problem that was observed is related to the querying service supported by a repository – it seems to be problematic for the *ANA core* (Ubuntu-linux based), in that environment, to deliver a message that is bigger than somewhere around 33 000 bytes, which implies the fact that very large query responds (more than 80 objects) could not be delivered back to the requesting *brick*.

Additionally the *top* [7] was used to measure the amount of memory used during the performance test. The measurements indicated that for storing 255 detected-incident objects and 255 alarm objects in each of the three virtual ANA nodes (this means a total amount of 255x3=765 alarm and 765 detected incident objects), a total amount of *9140 Kbyte* was consumed. This seems to be a good result, but should also be accepted as a warning that a garbage collection mechanism is needed to periodically swap out some of the alarm/incident information to an external database or to the file system of a node.

VII. CONCLUSIONS AND FURTHER WORK

The experiences gained in implementing UniFAFF for clean-slate type of network designs like ANA, though we have not fully implemented the framework, show that UniFAFF is indeed applicable in practice for the emerging self-managing networks, whether based on evolutionary approaches or clean-slate/revolutionary approaches. However, it is worthy mentioning that in order to implement the co-operative processes of information/knowledge sharing among network entities as defined by the UniFAFF framework, the "Requirements" put forward in the framework need to be strictly followed by developers of the individual functional entities of an autonomic node. The "Requirements" themselves as pointed out in the definition of the UniFAFF framework need to be further refined and further extended. The other important issue to note is that the issues proposed in the UniFAFF framework still require a lot of more research and development in order to have a complete implementation of the whole framework that allows extensive evaluations beyond the what we have achieved so far, meaning that a larger testbed emulating a large scale self-managing network would be required in order to fully evaluate UniFAFF in terms of information flow, resource consumption and network stability. Our further research work will involve further specification and design of the APIs of the Generic Architecture of UniFAFF we have not covered so far, and evaluate the performance and scalability of our design. We also aim at implementing UniFAFF for an evolutionary approach type of research towards future internet design in the context of the EU funded FP7 – EFIPSANS project [8] in order to draw experiences from both approaches.

REFERENCES

[1] The FCAPS management Framework: ITU-T Rec. M. 3400

[2] R. Chaparadza: UniFAFF: a Unified Framework for Implementing Autonomic Fault-Management and Failure-Detection for Self-Managing Networks, International Journal of Network Management, published by John Wiley & Sons, Copyright John Wiley & Sons 2008.

[3] N. Tcholtchev: Master of Science Thesis: Components and Mechanisms of Autonomic Fault-Management for Self-Managing Networks, submitted to the Technical University of Berlin, September 2008, to be publicly available soon (not yet publicly available as of 09 November, 2008).

[4] R. Chaparadza: Specification of the Failure-Detection and Fault-Management part of the ANA Architecture (v1), ANA Project Deliverable D.3.5v1, publicly available under: http://www.ana-project.org.

[5] C. Jelger and S. Schmid: ANA Blueprint (version 2), February 2008. ANA Deliverable D.1.4/5/6v2, available on http://www.ana-project.org

[6] M. Sifalakis: First Draft of the Functional Composition Framework, 15th February 2008, Deliverable D.2.4, available on http://www.ana-project.org

[7] The *top* man page: http://linux.die.net/man/1/top: as of date: 03.09.2008

[8] EC funded- FP7-EFIPSANS Project: http://efipsans.org/