A GENERIC CONCEPT FOR PROTOCOL DETECTION AND ITS IMPLEMENTATION FOR UMTS MONITORING

INA SCHIEFERDECKER, MARION SCHÜNEMANN, AIHONG YIN
Technical University Berlin, Telecommunication Networks Group, Str. des 17. Juni 135
10623 Berlin, Germany

PETER H. DEUSSEN, AXEL RENNOCH
Fraunhofer Institute FOKUS, CC TIP, Kaiserin-Augusta-Allee 31
10589 Berlin, Germany

This contribution presents a generic approach and implementation for protocol detection and traffic analysis (TA). Self-detection and auto-configuration facilities minimize the effort to manually configure a monitor or test device and should be an integral part for test automation. We elaborate our concept on the intelligent self-detecting features for a UMTS monitor focussing on interfaces, protocols and signaling scenarios in the context of the UMTS Terrestrial Radio Access Network (UTRAN).

INTRODUCTION

The acceptance and success of new telecommunication systems depends not only on the innovation and availability of new services for the end-customers, but also on the reliability of the underlying (network) infrastructure. Since today's market pressure tends to give less time for sufficient testing of products and networks, a fast and easy access to in-operation monitoring, fault detection, and traffic or call analysis is needed to analyse networks and network elements in experimental, field trial and production environments.

The operation of the UMTS infrastructure [1] demands the development of appropriate equipment to observe and analyse the functional behaviour of involved network elements. Passive monitoring of the inter-communication between UMTS nodes can be used for the identification and evaluation of network behaviour and protocols at selected interfaces, only if a sufficient level of information details is considered and an adequate part of the network infrastructure is included in the monitoring process. The number and complexity of UMTS procedures is big and requires a profound consideration of protocol data and dependencies. The development of an intelligent UMTS monitor must consider these circumstances for the provision of facilities for protocol and topology recognition.

Generic concepts for protocol detection will be discussed and used for the implementation of intelligent self-detecting and auto-configuring features and an efficient TA as part of a commercial UMTS monitor that is able to identify UTRAN interfaces [2]. A flexible software structure will allow ad-hoc changes and maintenance according to e.g. modified protocol standards, vendor-specifics or network operation requirements. We have investigated the details of UTRAN interfaces and protocols and architectural

requirements for self-detection and auto-configuration. The objective of protocol self-detection is that the monitor can automatically find out, which protocols are running on the interface or link to which it is connected. It is the basis for the monitor auto-configuration that frees the user from time-consuming complex configuration tasks allowing him to concentrate on the equipment and network analysis. For this purpose, traffic must be analysed for certain protocol patterns, which the monitor can use to unambiguously identify the corresponding protocols. For the protocol detection, the protocol stacks appearing at the UMTS interfaces Iu, Iur, and Iub have been considered.

For the realisation of the protocol and topology self-detection, some requirements have been identified. Self-detection bases on analysing existing network traffic, i.e. PDUs. It is also possible that channels change dynamically during the monitoring process (e.g. setup and release of transport channels), which requires a reconfiguration of the monitor. For these reasons, the self-detection process should be repeated in certain time intervals during an ongoing monitoring process.

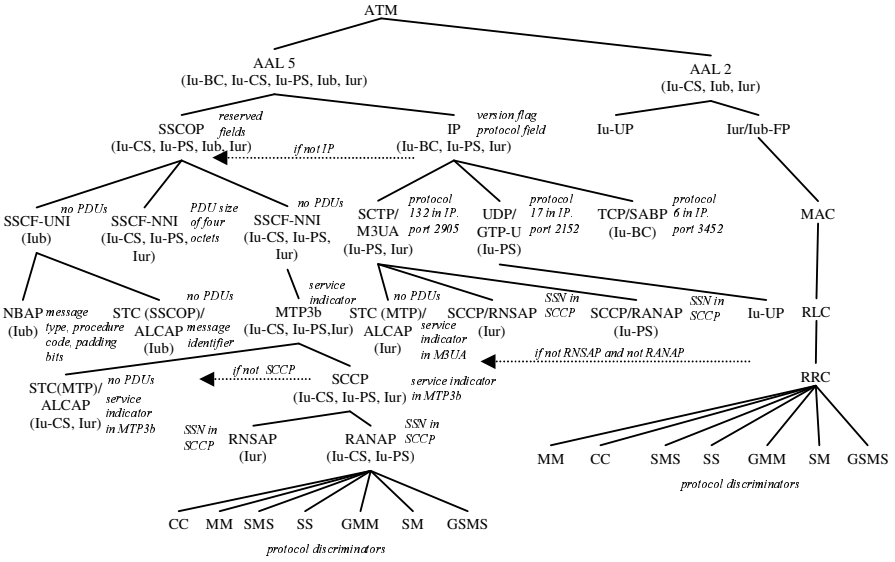

Figure 1. UTRAN Protocol tree.

In course of the self-detection process, captured protocol information has to be analysed and validated against protocol identification patterns. In particular, we made use of: Protocol Discriminator, Subsystem Number, Port Number, Length Information, Checksum (CRC), Message Types, Padding bytes/bits, and dedicated parameter value ranges. Considering the different protocol stacks existing within UTRAN, a combined tree representation of these protocol stacks as illustrated in Figure 1 has been developed

and used. It enables a new view on the protocol dependencies and illustrates where protocol detection decisions have to be made. Considering the protocol tree for the UTRAN, one can see that ATM is located in the root of this tree. The protocol self-detection has its starting point at the root of the protocol tree, i.e. it starts with the capturing and analysis of ATM cells.

For the identification of protocols and protocol stacks, an algorithm that traverses the protocol tree must be used. We distinguish two approaches: (1) Top-down approach: In this approach, single layer templates are used to detect a certain protocol, starting at the root of the tree. For the next and all further tree levels, additional protocol templates are added to the already constructed protocol stack. (2) Bottom-up approach: Full-stack templates are defined in advance for each tree branch. These complete templates are applied successively to identify the protocol stack in use as a whole.

GENERIC CONCEPT

The architecture for the realisation of the protocol self-detection consists of different components which are illustrated in Figure 2 and are described in the following. The initialisation of these components during a start-up period is done by a control and configuration component. The traffic analysis (TA) components contain predefined protocol information, i.e. protocol templates, protocol trees, and protocol patterns.

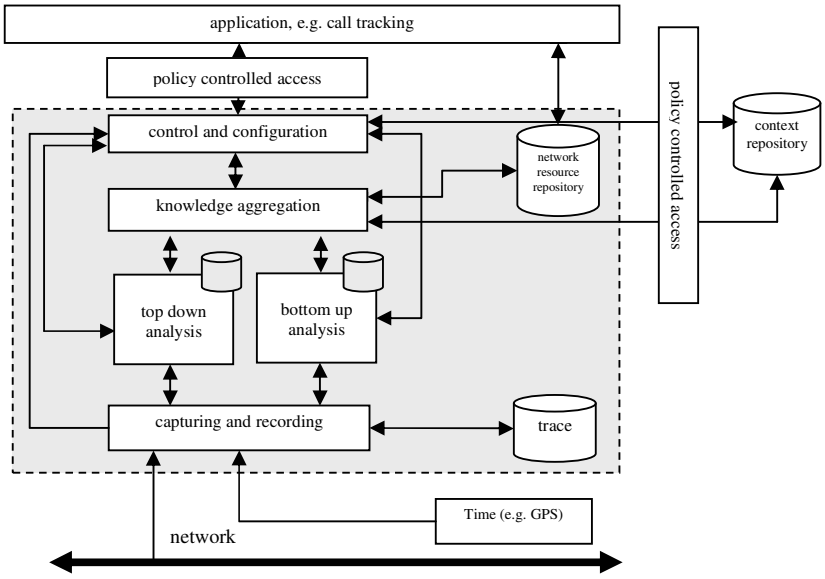

Figure 2. Self-detection architecture.

The capturing and recording component is connected to a network link to monitor the data traffic. An integral part of the self-detecting architecture is the knowledge aggregation component. It controls the analysis components and receives the results of their analyses. The interpretation and combination of the observations as well as the result evaluation is done by this component.

In the following we concentrate on the TA implementation concept, since it is the central part of the protocol self-detection. Concerning autoconfiguration, it is influenced by two premises: (1) Protocol detection code should be reusable in different context, e.g. the implementation of a specific protocol pattern identification should be realised only once, even it can appear in different protocol trees or at multiple tree positions (like ALCAP PDU identification at NNI or UNI). One or more hard coded decisions or decoder calls may be combined in one module that checks an input frame whether it fits with given templates, i.e. the assumptions to be tested. (2) Furthermore, modules as defined above may be separated into a sequence of modules, i.e. a hierarchy with multiple levels is introduced. Modules at a lower level n may call a module at a higher level $n + 1$ if its analysis assumption (i.e. an input frame belongs to protocol layer n) has been validated.

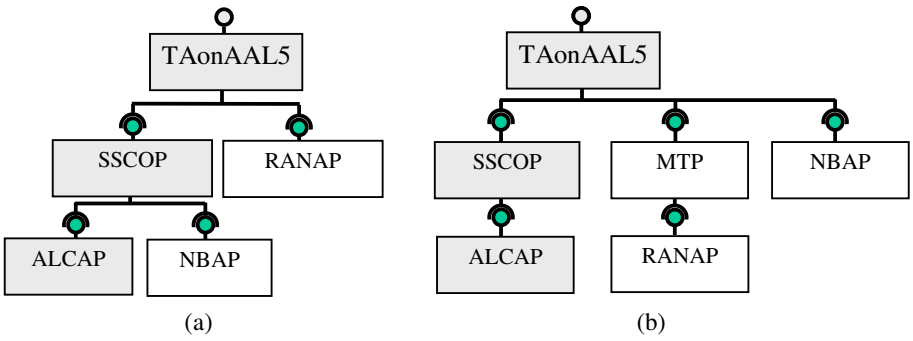

Figure 3. Flexible TA strategies.

Such hierarchical structures follow the protocol tree and detection algorithm, which needs to be decomposed in meaningful parts that may be reused. Engineers are free in their strategies and preferences on how to define the scope of a single module. They have to define an approach, which allows enough flexibility to change the module priorities/ordering. Figure 3 gives two different module structures that address the same target protocols (ALCAP, NBAP, RANAP). If you suppose that at one selected level a module on the right has a higher priority than its left neighbours then you can read the structures as follows: In variant (a) NBAP (over SSCOP) detection will be considered later than a RANAP (over MTP) check and after a SSCOP detection (shared with the ALCAP module). A test operator may wish e.g. to give the NBAP check a higher priority and split the RANAP and MTP check as illustrated in variant (b). A shift of the NBAP to

the most right position of the lower level should give such effect, but the removal of the RANAP detection from level 1 to level 2 requires a module split that has to be considered earlier during the software implementation phase. Ad-hoc changes require a sufficient degree of decomposition. On the other hand a very large amount of modules with small scopes may increase overhead and reduce the efficiency.

The introduction of a flexible tree configuration requires a description of the module structures and some means to select and load the preferred strategy during an initialisation phase. Using e.g. XML descriptions may be comfortable to select (or exchange) some modules (and (re)define the relationships) of dedicated pre-defined (i.e. implemented) types. Such modules can be stored as part of a software library (e.g. object or .dll files). They may be subject to further specialisation with e.g. customer specific constraints (template parameter). A set of native types could also be implemented in modules to allow the use of basic behaviour like if-then-else constructs or logical operations.

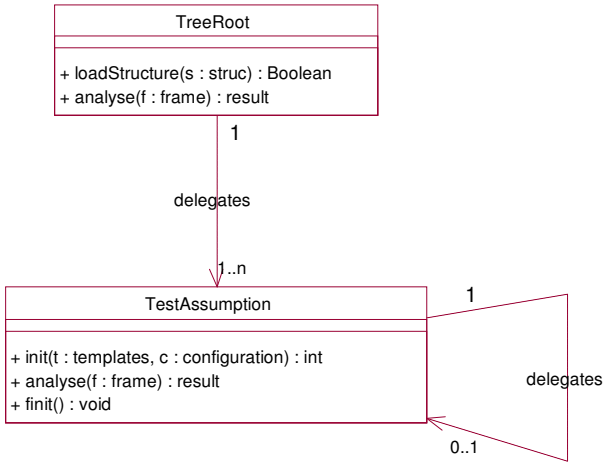

Figure 4. Generic TA: class diagram.

In Figure 4 we illustrate basic software classes required to implement the TA: The TestAssumption objects get a request to analyse a frame either from a TreeRoot object or another instance of TestAssumption (illustrated with delegates associations), i.e. - as introduced before - TestAssumption objects can delegate a part of a result analysis to instances on further protocol tree levels. It is obvious that the TestAssumption instances must be initialised (init operation) according to their particular position in the tree in alignment with the expected type of input frame. In addition to the basic configuration parameter, specific templates defined for usage within a module are input to TestAssumption as part of the init operation. On the other hand the result types (values)

have to be defined: Each instance may return an unknown value if the input frame has not matched the pattern of the assumption. If input frame and template fit together, an identifier for the detected protocol will be returned. The result(s) from higher levels may be combined with the identifier of the calling (parent). A TestAssumption instance, e.g. a MTP_SCCP module that receives an indication for RANAP detection, may return MTP_SCCP.RANAP to its calling instance.

We do not put restrictions to the dedicated tasks of the TestAssumption modules in the generic concept. A TestAssumption instance may call further (or all) of its associated objects even if one instance has already returned a positive detection result. Thus a double (or multiple) match may happen, if the considered assumptions (or their implementations) do not exclude completely each other. Such cases may be due to e.g. a combination of different interfaces on the same line or customer specific modifications. Anyway, such detection results should be returned to the end-user with appropriate conflict information. Alternatively, the TestAssumption object may have implemented conflict resolution mechanisms (based e.g. on probabilities) that will resolve the conflict locally and return a unique result (probably accompanied by some comment).

IMPLEMENTATION

Starting from the selected protocol patterns and the protocol tree, an algorithm has been developed that follows the top-down approach. Considering the protocol tree, we can distinguish between protocols available in the protocol tree as a tree node and such available as a tree leaf. Figure 5 gives an overview on some implemented software classes. In the IP branch, there are two classes implemented for protocols tree nodes: TAIuxIP which addresses a combined detection of IP and the next higher protocols, and TAIuxM3UA. For each tree node protocol of the SSCOP branch, two further classes have been foreseen: TAIuxOnSSCOP and TAIuxMTP3b. A common class, called TAIuxSCCP, is used for the SCCP detection both over SSCOP and IP. The classes for the tree node protocols have been specified to realise the analysis of more detailed protocol information, e.g. SI of MTP3b, SI of M3UA or SSN of SCCP, and also to initiate and decide the analysis of protocols on the next tree level. For the tree leaf protocols, one common class has been specified, the TAProtDec class. Using this class, a protocol frame can be completely decoded.

An implementation of the TA needs to be developed carefully because of the following major product quality issues that are relevant: performance, maintainability, and reliability. According to the requirements for self-detection stated above, the question on how to define and implement required protocol identification patterns plays an important role and has to be answered early in the development process. Existing approaches may focus on an optimisation of either the TA performance or its maintainability: (A) A so-called hard coding variant means that any protocol pattern should be coded and used as simple as possible (to avoid any time overhead). (B) A decoder tool variant will use a

suitable repository and access to the protocol structures, codec rules, protocol (stack) dependencies etc., i.e. details like field positions and pattern length may be excluded from the particular TA implementations. Due to practical considerations, a combination of (A) and (B) that benefits from a decoder tool for selected decisions only (e.g. if complex codec rules are involved) is preferred and has been used in our implementation.

Our implementation became part of a commercial UMTS monitor and has been applied to UTRAN interfaces. Figure 6 provides a screenshot of the user GUI that identifies RANAP (over SSCOP). The performance of our demonstrator has been measured. A direct comparison with previous measurement of a hard coded implementation variant seems to be difficult, but it is obvious that the protocol detection time is increased. For this fact, two main reasons apply: (1) The whole software structure has been changed to realise a more generic concept. This includes e.g. more function calls during frame decoding and consequently an increase of the decoding time. (2) Due to the consideration of additional interfaces by the auto-configuration more protocols have to be analysed. The algorithm starts with a detection of MTP3b messages, which increases the decoding time for ALCAP and NBAP. The realisation of a reliable protocol detection concept in the demonstrator requires an ALCAP and NBAP decoding attempt before a decision can be made.

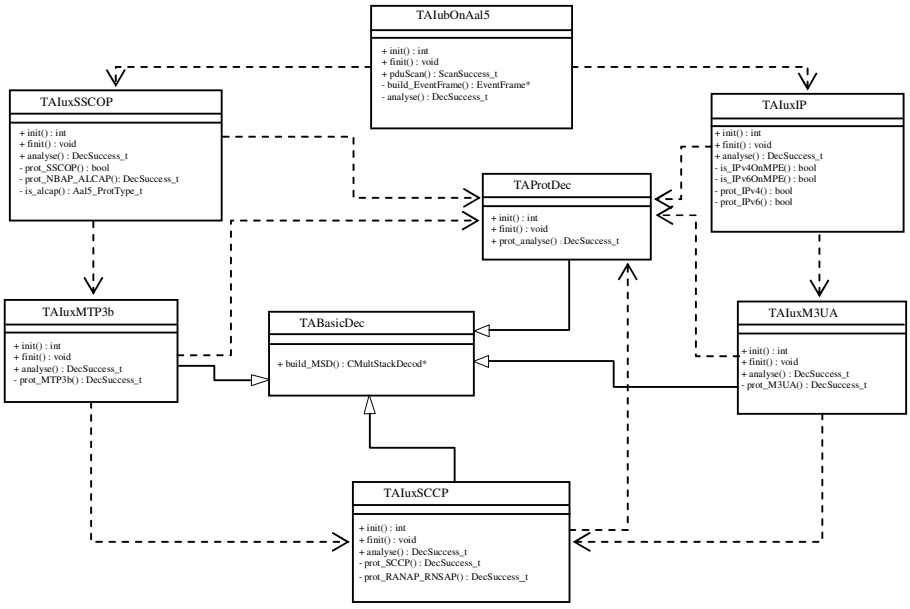

Figure 5. Software Classes (AAL5 branch).

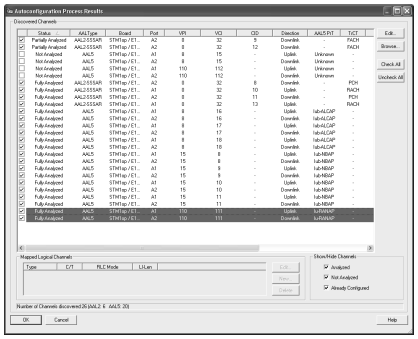

Figure 6. RANAP detection (screenshot).

OUTLOOK AND FURTHER RESEARCH

The generic concept for self-detecting traffic analysis as a basis for monitor auto-configuration described in this document can be characterised and compared to the elements of a test program consisting of data types, values (templates), and some basic behaviour that could be found in test notations like TTCN-3 [3]. It should be possible to implement the detection algorithm at an abstract level and to use automatic tool support for compilation and execution of such test scenarios. Although readability of monitoring and testing data templates and protocol procedures based on TTCN-3 will increase, it is assumed that performance requirements might be failed. At this point a performance case study is needed to have an insight into the conditions and possibilities for the use of advanced test notations for monitoring purposes.

Acknowledgements
This work has been done within the Trillian-T project that has been supported by the European Regional Development Fund. We like to thank our colleagues at Tektronix Berlin, in particular Hans-Werner Arweiler, Stephan Klug, Birgit Kähler, Hubertus Toenne and Andreas Vehse for detailed technical discussions and support.

REFERENCES

[1] UMTS Networks Architecture, Mobility and Services, Heikki Kaaranen, Ari Ahtiainen, Lauri Laitinen, Siamäk Naghian, and Valtteri Niemi, 2001,Wiley.

[2] Technical Specification 25.401 V3.10.0, UTRAN Overall Description (Release 1999), 3rd Generation Partnership Project (3GPP), June, 2002.

[3] ES 201 873-1 V2.0.0: The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, European Telecommunications Standards Institute (ETSI), October 2001.