LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

NEW TTCN INTERFACES FOR WCDMA BASE STATION TESTING

Diplomityön aihe on hyväksytty Lappeenrannan teknillisen korkeakoulun tietotekniikan osaston osastoneuvoston kokouksessa 22.1.2003.

Työn tarkastajana toimi professori Jari Porras ja tarkastajana sekä ohjaajana DI José Manuel Tapia.

Lappeenrannassa 8.2.2005

Timo Särkikoski
Korpisuonkatu 1 D
53850 Lappeenranta

ABSTRACT

Author: Särkikoski, Timo

Subject: **New TTCN interfaces for WCDMA base station testing**

Department: Information technology

Year: 2005

Master's Thesis. Lappeenranta University of Technology. 77 pages, 38 figures and 5 tables.

Examiners: Professor Jari Porras and MSc José Manuel Tapia

Keywords: TTCN, WCDMA, automated testing

WCDMA base station (Node B) is located in the radio network part of the UMTS. Node B is an important network element providing mobile users connectivity to the network. Telecom software (TCOM SW) in the Node B provides a large part of Node B's functionality. A lot of effort is put in TCOM SW testing to ensure correct functionality and high quality. System component testing is a testing phase where a building block (system component, in this thesis TCOM SW) of a system (Node B) is tested before integrating it to other system components. A test tool is needed to provide a test environment and implementation of test cases. Node B TTCN Tester (the tester) is a tool for testing Node B software. TTCN test notation is used to define test cases, which are executed against Node B by using the tester. For system component testing of TCOM SW new interfaces were added to the tester platform for simulating ATM software, WPA unit and WTR unit of Node B. In this thesis TTCN test cases were implemented using the new tester interfaces. The test cases made TCOM SW system component testing independent of Node B ATM software, WPA and WTR. In addition testing of TCOM SW functionality in ATM, WPA and WTR interfaces was automated. The implemented test cases were executed against Node B with successful results. The new test cases and TTCN interfaces worked well and increased efficiency of testing.

TIIVISTELMÄ

Tekijä: Särkikoski, Timo

Aihe: **Uudet TTCN rajapinnat WCDMA tukiaseman testaukseen**

Osasto: Tietotekniikan osasto

Vuosi: 2005

Diplomityö. Lappeenrannan teknillinen yliopisto. 77 sivua, 38 kuvaa and 5 taulukkoa.

Tarkastajat: Professori Jari Porras ja DI José Manuel Tapia

Hakusanat: TTCN, WCDMA, testausautomaatio

WCDMA tukiasema (Node B) on osa UMTS-järjestelmän radioverkkoa. Node B on tärkeä verkkoelementti, jonka tarkoituksena on yhdistää mobiilikäyttäjät verkkoon. Telecom –ohjelmisto (TCOM SW) on vastuussa suuresta osasta Node B:n toiminnallisuutta. TCOM SW:n testaukseen käytetään paljon resursseja, jotta ohjelmiston oikeasta toiminnasta ja laadusta voidaan varmistua. System component testing on testausvaihe, jossa järjestelmän (Node B) osa (system component, tässä diplomityössä TCOM SW) testataan ennen sen integroimista muuhun järjestelmään. Tähän tarvitaan testityökalu ja testitapausten toteutus. Node B TTCN Tester (testeri) on työkalu, jota käytetään Node B:n ohjelmiston testauksessa. Testitapaukset toteutetaan TTCN-testinotaatiota käyttäen ja testataan testerin avulla. TCOM SW:n system component –testausvaihetta varten testeriin lisättiin uudet rajapinnat, joiden avulla voidaan simuloita Node B:n ATM-ohjelmistoa sekä WPA- ja WTR-yksiköitä. Tässä diplomityössä toteuttiin TTCN testitapaukset uusille rajapinnoille. Testitapaukset tekivät TCOM SW system component –testausvaiheen riippumattomaksi Node B:n ATM-ohjelmistosta sekä WPA- ja WTR-yksiköistä. Lisäksi TCOM SW:n toiminnan testaus näissä rajapinnoissa voidaan tästä lähtien tehdä automaattisesti. Testitapauksien toiminta varmistettiin testeriä käyttäen. Tulokset olivat hyviä, uudet testitapaukset ja TTCN rajapinnat toimivat oikein lisäten testauksen tehokkuutta.

PREFACE

This thesis has been done in NetHawk Oyj Lappeenranta office as part of the NBTM project. I want to thank José Manuel Tapia from Nokia Networks for reviewing my work and giving valuable comments. This thesis would not have been possible without his open and positive attitude. I would also like to thank professor Jari Porras at the Lappeenranta University of Technology for reviewing the thesis and positively urging me to graduate. Special thanks go to my parents Hannu and Tuija for always supporting me in my studies. Finally I thank my dear Outi for encouraging me during the writing process.

In Lappeenranta, 8th of February 2005

Timo Särkikoski

TABLE OF CONTENTS

ABBREVIATIONS

3G	3rd Generation mobile networks
3GPP	3rd Generation Partnership Project
AAL	ATM Adaptation Layer
ALCAP	Access Link Control Application Part
ASN.1	Abstract Syntax Notation 1
ASP	Abstract Service Primitive
ATM	Asynchronous Transfer Mode
ATS	Abstract Test Suite
BB	Baseband
BCH	Broadcast Channel
BTS	Base Transceiver Station, in UMTS BTS is called Node B
CCITT	Comité Consultatif International Téléphonique et Télégraphique
CM	Coordination Message
CPCH	Common Packet Channel
CS	Convergence Sublayer
DCH	Dedicated Channel
DSC	Data, Signaling and Control
DSCH	Downlink Shared Channel
DSP	Digital Signal Processor
ETS	Executable Test Suite
ETSEU	Executable Test Suite Execution Unit
ETSI	European Telecommunications Standards Institute
FACH	Forward Access Channel
FDD	Frequency Division Duplex
FIFO	First In First Out
FP	Frame Protocol
HS-DSCH	High Speed Downlink Shared Channel
HW	Hardware
IE	Information Element
IEC	International Electrotechnical Committee

IP	Internet Protocol
ISO	International Standards Organization
ITEX	Interactive TTCN Editor and eXecutor
ITU-T	International Telecommunication Union – Telecommunications standardization sector
Iu	Interface between UTRAN and core network
Iub	Interface between Node B and RNC
Iur	Interface between two RNCs
MCP	Master Coordination Point
MTC	Master Test Component
NBAP	Node B Application Part
Node B	Base station in UMTS network
O&M	Operation and Maintenance
PCH	Paging Channel
PCO	Point of Control and Observation
PDU	Protocol Data Unit
PEU	Protocol Execution Unit
PTC	Parallel Test Component
R&D	Research and Development
RAB	Radio Access Bearer
RACH	Random Access Channel
RAN	Radio Access Network
RF	Radio Frequency
RFM	Radio Frequency Module
RNC	Radio Network Controller
RNS	Radio Network Subsystem
RP	Reference Point
SAAL	Signalling AAL
SAP	Service Access Point
SAR	Segmentation and Reassembly Sublayer
SSFN-UNI	Service Specific Coordination Function – User Network Interface
SSCOP	Service Specific Connection Oriented Protocol

SSSAR	Service Specific Segmentation and Reassembly sublayer
STC	Signalling Transport Converter
SUT	System Under Test
SW	Software
TASSU	Testing Analyser and Simulator for Systems and Units
TASSU Iub	Iub interface using TASSU connection instead of ATM
TCOM SW	Telecom software
TCP	Transmission Control Protocol
TDD	Time Division Duplex
TR	TASSU Router
TTCN	Tree and Tabular Combined Notation (TTCN-1 and TTCN-2) Testing and Test Control Notation (TTCN-3)
TTS	TASSU Test Support
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UNI	User-Network Interface
USCH	Uplink Shared Channel
UTRAN	Universal Terrestrial Radio Access Network
Uu	Interface between Node B and UE
VCI	Virtual Channel Identifier
VPI	Virtual Path Identifier
WAM	Wideband Application Manager
WCDMA	Wideband Code Division Multiple Access
WPA	Wideband Power Amplifier
WSP	Wideband Signal Processor
WTR	Wideband Transmitter and Receiver

1 INTRODUCTION

Universal Mobile Telecommunications System (UMTS) offers new features and higher data rates compared to Global System for Mobile Communications (GSM). This requires complex network elements, such as Wideband Code Division Multiple Access (WCDMA) base stations. Development of base station software is a demanding task. There are a lot of system components (subsystems) that need to work together to produce required functionality. Each system component must be thoroughly tested before putting it together with other components. System component testing is done for this purpose. Telecom software (TCOM SW) is an important part of base station functionality. System component testing of TCOM SW requires tools that can simulate the environment in which TCOM SW is used in the final product. Producing and maintaining these test tools requires significant effort, but the effort is worth spending. The earlier errors are detected, the easier and cheaper they are to correct. Tree and Tabular Combined Notation (TTCN) language and a TTCN tester tool are used in testing. To enable more thorough testing of TCOM SW the TTCN tester's simulation capabilities are improved by adding new TTCN interfaces.

The scope of this thesis is to develop new TTCN interfaces for system component testing of WCDMA base station TCOM SW. The new interfaces are used to simulate ATM software, WPA unit and WTR unit. The first task in the thesis is to select the test cases to be implemented in TTCN. Next the test cases are implemented using TTCN language. The last task is to verify that the test cases and the new interfaces work as expected by running them against WCDMA base station using TTCN tester.

This thesis focuses on producing the first versions of the TTCN test cases that use the new ATM, WPA and WTR interfaces. The first release consists of test cases, which are needed to set up the basic services in the WCDMA base station, such as cells, common transport channels and radio links. When the test cases are ready they can be used as a reference and their TTCN code can be reused in the future development.

1.1 NBTM Project

This thesis has been done in Node B Tester Maintenance (NBTM), which is a Nokia Networks project developing and maintaining TTCN tester for WCDMA base station (Node B). TTCN test suite maintenance and most of the test case development is done as a subcontracting project in the Lappeenranta office of NetHawk Oyj. The end users of the test cases produced in this thesis will be the TCOM SW developers at Nokia Networks.

1.2 Structure of the Thesis

The first chapter introduces the thesis and the project. The second chapter explains why software testing is done and presents methods how testing can be used to reduce errors in the software. Different aspects of test automation are also presented. The third chapter describes UMTS, the telecommunications system containing the WCDMA base station. The fourth chapter takes a closer look at the WCDMA base station and the components important to this thesis. The fifth chapter presents TTCN test notation and Nokia Node B TTCN tester, which are the main tools used in the thesis. The sixth chapter describes the practical part of the thesis, the TTCN implementation of the test cases with the ATM, WPA and WTR interfaces. The seventh chapter draws conclusions of the thesis.

2 Software testing

[1 p. 77] defines software testing as a process of "executing a software system to determine whether it matches its specification and executes in its intended environment". In addition to verifying correct functionality the purpose of testing is to reveal possible errors before a product is delivered to a customer. Eliminating all errors is very difficult, but good testing practices can improve software quality a lot. Customer satisfaction is important in competitive markets. A product that does not do what it is promised to do or is not usable because of errors will make customers unhappy, which makes them find another supplier for their needs. Testing is an essential part of software development process from the beginning to the end. The earlier errors are detected the easier and cheaper they are to correct. Table 1 shows an example of magnitudes of error removal cost in different development phases.

Phase	Cost
Definition	$1
High-Level Design	$2
Low-Level Design	$5
Coding	$10
Component Testing	$15
Integration Testing	$22
System Testing	$50
Post-Delivery	$100+

Table 1: Error removal cost over product development life cycle [2 p. 8]

2.1 Testing strategies

Testing strategy includes the way test cases are designed and described, how testing progress is reported, how tests are grouped and assigned, and how decisions are made regarding what tests need to run and when. Different models of software development put different weight to software testing. Traditional waterfall model (Figure 1) shows testing as the last part of the development process after requirements analysis, design and implementation. With this model it is easy to underestimate the importance of

testing. Instead of viewing testing as a way to improve product quality, testing could be seen as the last obstacle before product can be released.

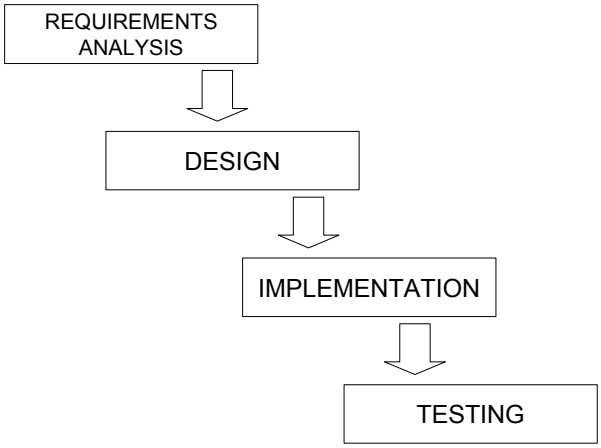

Figure 1: Waterfall model

V-model shown in Figure 2 describes development of complex software more accurately than the waterfall model. Instead of showing testing as a single activity, different testing phases are put in relation with software development phases. They can be seen as parallel activities, software development activities are shown on the left side and corresponding testing activities are shown on the right side. The number of phases in development and testing solely depends on the engineering process followed by the developers. The phases used in Figure 2 are adapted from [3 p. 9]. By following V-model it is likely that testing preparations are given enough time and resources, and testing in general will be given thought already in the beginning of product development. This way testing can improve quality instead of just measuring it.

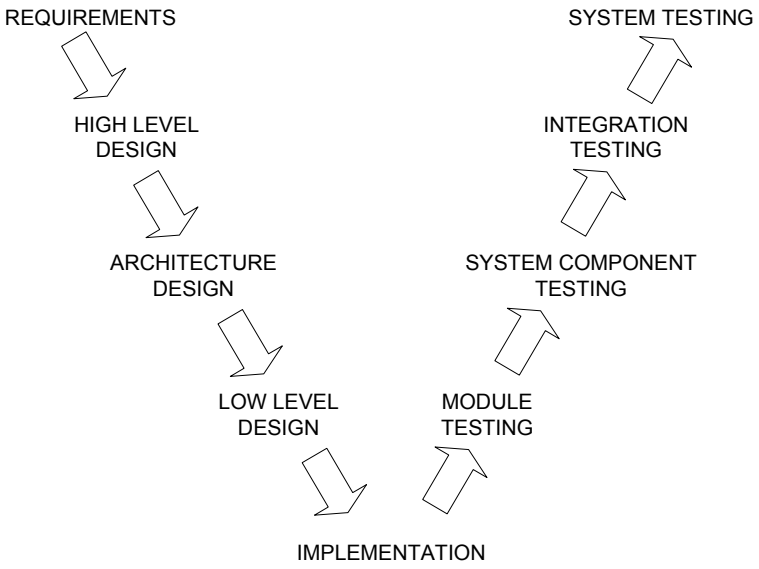

Figure 2: V-model

2.2 Testing phases

V-model divides testing into different phases according to product development phases. This section gives an overview about the testing phases.

Module testing (also called component or unit testing) is targeted on software modules. The definition of software module varies depending on the product characteristics and purpose of testing. Generally a module is the smallest entity that can be tested separately. The main purpose of module testing is to verify that a module meets its design specifications and to find errors that typically occur when source code is written. It is also verified that the module is ready for system component testing. Usually software engineer responsible for the code performs the testing since he has an intimate knowledge of the module source code and is the best person to find faulty parts quickly. Testing technique is white box oriented. Tested modules can be so limited in functionality that they need supporting software, drivers and stubs, to enable tests (Figure 3). A driver is a program that passes data to the module and displays possible results. Stubs are services that the module needs to fulfil requests of the driver program. It is very important that module testing is done properly, because correcting errors is much more difficult and expensive in the later part of the development process. [1 p. 77] [4 p. 487]

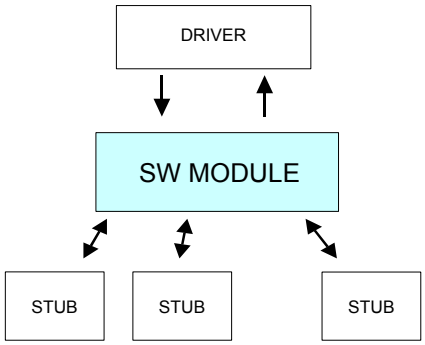

Figure 3: Driver and stubs

System component testing is targeted on testing system components, in other words the architecture blocks of the product. A complex product consists of several subcomponents, each providing a part of functionality needed in the complete product. A system component in itself can have complex functionality and contain large amount of source code. This is the case for example with Nokia WCDMA base station, in which many system components are needed to produce required functionality. The system component related to this thesis, Telecom software, has an important role in operation of the product. In this testing phase it is verified that the system component provides the required functionality. Testing is white box oriented and carried out by the system component developers. [3 p. 9]

Integration testing concentrates on verifying that system components work with each other, and that together they deliver specified functionality. Testing concentrates on system component interfaces and inter-component communication. In a single test a subset of all the components is tested. [1 p. 77]

System testing tests a set of components that comprise a deliverable product. Software is tested in its final environment, with all system elements in place. It is verified that the system works in real operating environment, has all the specified functionality and unwanted functionality does not appear. [3 p. 9]

2.3 Testing techniques

This section describes common techniques used in software testing.

2.3.1 White box testing

White box testing (also called code based testing) is a technique where *knowledge about internal functionality of software is used*. Software is seen as a "glass box" (Figure 4). Test cases are derived from software module design specifications. They test source code implementation, for example data structures, flow control and memory handling. [5 p. 80]

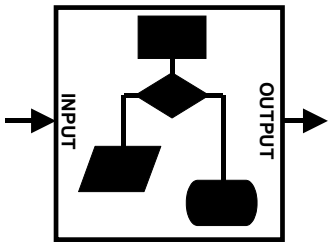

Figure 4: White box testing

When a test case is run, it will execute a part of software's internal logic. Ideally the internal logic should be tested completely, but with complex software this would require a large number of test cases. White box methods try to find errors in the logic and verify that test coverage is sufficient. According to [2 p. 239] white box methods provide following services:

- They guarantee that all independent paths within a module have been exercised at least once.
- They exercise all logical decisions on their true and false sides.
- They execute all loops at their boundaries and within their operational bounds.
- They exercise internal data structures to ensure their validity.

Table 2 presents a concise list of white box testing techniques.

Technique	Description
Fault insertion	Errors are forced and behaviour of the code is observed. This is a good way to test simulate real error situations.
Error handling	It is assumed that not all errors are possible to remove from the source code. By making sure that error handling in the software works properly it is less likely that unexpected errors cause big trouble.
Coverage analysis	*Statement* coverage executes each statement at least once. *Decision* coverage executes each decision at least once. *Condition* coverage executes each path in the logic flow at least once (path coverage).
Memory leak	Find instances where application is not handling memory allocation correct.

Table 2: White box techniques [2 p. 239-243]

2.3.2 Black box testing

Black box testing (also called specification based testing) is a technique where *internal operation of software is not taken into account* and functionality is tested using specified interfaces. Software is seen as a black box, which has interfaces for input and output (Figure 5). In this method it is checked that a given input results in pre-defined output, what happens in between is not relevant to test case execution. Test cases are derived from requirements specifications.

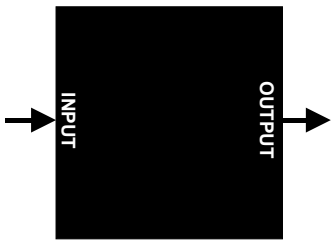

Figure 5: Black box testing

Black box methods try to find errors in software functionality, interfaces, usability, performance and security. [2 p. 245]

Table 3 presents a concise list of black box testing techniques.

Technique	Description
Boundary value analysis	Data used in software usually has defined set of values. Value boundaries are identified. Tests are done with values, which are lower than, higher than and on the boundary of accepted values.
System testing	Functional, regression, security, stress, performance, usability, random, data integrity, conversion, backup and recoverability, configuration, operational readiness, user acceptance, alpha / beta testing.
Equivalence partitioning	Assume that not all input variations can be covered. Divide inputs in equivalence classes, and use a subset of each class. For example classes could be made for values, which are within accepted range and for values, which are outside of accepted range.

Table 3: Black box techniques [2 p. 245-247]

2.4 Automated testing

Test automation takes advantage of tools, which can execute tests without or with little human assistance. Automated testing brings many benefits to the software development process when it is applied correctly. It also has drawbacks, which must be taken into account.

In highly competitive software markets different products compete with features. New versions of product with new features must be brought to market often to keep customers happy and to prevent them from changing their product supplier. Rapid Application Development (RAD) tools enable quick product creation and make addition of new features easier. Test automation is an essential tool to keep high standard of quality while keeping development cycles short. Software is often built incrementally,

starting with basic functionality to get the product out on the market quickly. New features are added later according to customer needs. When new functionality is added, it must be verified that existing functionality is not affected, which means repeating same test procedures often. Significant savings in time, effort and test quality can be made if the testing process can be automated. [2 p. 3-4]

2.4.1 Benefits

The most obvious benefit of automated testing is that it **reduces possibility of human errors** during test case execution. When a test case has been automated and is proved to be working correctly, it will continue working correctly as long as the test environment and the system under test conform to test case specifications. Automatic test will run always the same way, it will not suffer from lack of concentration, which could be a problem with a human test engineer. In another words the **test case runs will be consistent**.

Tests can be run faster. When a human tester runs test cases, he will interact with the system under test. Interaction could take place for example by selecting and sending messages from a test tool and observing reaction to those messages. The test case specification could define a long list of message exchanges, which would have to be followed closely to successfully run the test. This kind of work requires concentration and cannot be carried out in a hurry. The longer and more complex the test case sequences are, the bigger difference in execution time will be between automatic and human tester. With automatic tester long and complex **tests can be run over and over again.** For a human tester this would require a lot of patience and very good ability to concentrate.

When test case has been automated, the test **engineer can concentrate on overall behavior of the system under test** rather than having to meticulously follow the test instructions. More rigorous tests can be done when automatic tester takes care of the labor intensive part and the human tester is freed to observe the test case progression.

In long term automatic testing **could bring savings in testing costs**. This is the case especially with iterative software development, when a product is developed in cycles. Building an automatic test system requires a lot of effort in the beginning, but after it has been established and if it is reducing human testing effort, it could save money. For example, software product could be developed in three consecutive releases A, B and C, each adding new features to previous releases. If product testing is automated for release A, it is likely that the cost of creating the automatic test system far exceeds the savings that the automation brings in that phase. However in the future releases B and C the automatic test system is already in place and is more likely to reduce testing costs.

2.4.2 Drawbacks

Automated testing **requires more resources than manual testing**. The effort required to develop and maintain an automatic test system is easily underestimated. **Automated testing does not detect errors that human test engineer would**. When a test case is automated, it will work the same way every time it is run. It takes into account only the kind of behavior it has been instructed about. Human tester is not restricted to the test case specification and will notice if behavior of the system under test is somehow strange. If testing generates an unexpected error, which has not been anticipated in test case specification, human tester is likely to notice it, but automatic script is not. Human intuition can pick up errors that automatic test scripts will never find. In this sense running the tests always the same way can be seen as a drawback for automatic testing, since running them with a little variance (which can happen with human testers) could reveal new errors. [6 p. 4] [7 p. 17]

2.4.3 Requirements

Test automation requires careful planning. Several questions must be answered before big actions towards automated testing are made:

- What is going to be automated?
- How it is done? How big is the effort?
- How are the test scripts and system maintained?

- What the cost benefits will be?

First it must be clarified *which test cases are going to be automated*. Quick answer would be to automate everything that possibly can be automated. A better strategy would be to identify those test cases that are easiest to automate and need to be tested most often. Required test environment of the test cases should be similar, so that the same test environment could be used for all of them. It could be that some tests are easier to do manually than to develop test environment for an automatic test tool.

Maybe the most crucial decision comes when it is time to *select the test tools*. As each product has its own characteristics, it is rarely possible to purchase an off the self testing tool and immediately start automatic test execution. These kinds of tools do exist for black box testing of products using standardized interfaces, such as telecommunication products. Even if the tool would use well known interfaces the way the test cases are defined might differ between tools. Selecting one means that the test cases must be scripted using methods specific to that tool. When a large number of test case scripts have been developed, it is not easy to change the test tool anymore. Using the existing test scripts directly with other tools is most likely impossible and modifying the scripts would be a big effort. Situation is more complicated when module testing process is automated. Internal interfaces of the product are less likely to conform to any standards, thus dedicated testing tools must be developed, or commercial test tools must be adapted to enable communication with the system under test.

When the test tools have been selected and it is clear what test cases will be automated, the *test script development* can start. The amount of effort should not be underestimated. Test script development is software development, and similar engineering practices should be used for test case development as for developing the software itself such as planning, documenting and reviewing. Developing the test scripts requires programming skills and knowledge of the system that will be tested. Skilled resources should be available and reserved when testing is planned. In addition to the test script writing, the test system which executes the tests must be developed. This means

creating software and hardware, which enable communication between the system under test and the test tool.

Maintenance of the test scripts and test environment is important and it should also be carefully thought when the decisions are made. As the product evolves, new features are added and old features are changed. These changes reflect to the test cases. The test environment must be kept up to date, so that the test cases can still be executed. The test scripts must be updated to work according to new product behavior. Software and test case developers should have good communication, so that changes in requirement specifications translate to changes in test case specifications as soon as possible. When the product features and the test cases do not match, there is not much benefit to be expected from testing. Important faults might not be detected or irrelevant fault reports could be generated. The latter will not enhance the cooperation between software developers and test engineers.

Automated testing does not necessarily reduce testing costs, because development and maintenance of automatic test system and test scripts is more or less costly. Possible savings will come in long term. According to [8 p. 23] investment in test automation makes sense if the costs of automation is less than the potential costs of:

- Supporting defective software in production.
- Engineering maintenance releases to fix problems in production.
- Losing business due to customer dissatisfaction.

2.4.4 Automated testing of WCDMA base station software

Software development of a WCDMA base station (Node B) follows an evolutionary delivery model. Different parts of the Node B software are developed at the same time. Development of the next version starts before current version is ready. In new versions features are added and errors detected in previous versions are fixed.

Automated testing is used in Node B software development. Nokia Node B TTCN Tester enables test automation and test script creation using TTCN language. The largest effort of creating the tester and the basic test cases has already been done. The longer the tester is used in testing of Node B software, the better investment it will be. Maintenance and continuous development of the tester are important to keep the tool updated and to add value to the test engineers. The tester has been used for testing several versions of Node B software. [9 p. 7]

3 UMTS

This chapter gives an overview of the Universal Mobile Telecommunications System (UMTS). More detail is given to UMTS Terrestrial Radio Access Network (UTRAN), which is the most important part of UMTS related to this Thesis.

3.1 UMTS Architecture

UMTS is a 3rd generation mobile network specified by 3rd Generation Partnership Project (3GPP). UMTS can be divided into three parts, as shown in Figure 6: Core Network (CN), UMTS Terrestrial Radio Access Network (UTRAN) and User Equipment (UE). CN is the basic platform for all communication services provided to the UMTS subscribers. UTRAN provides radio communication between Core Network and User Equipment. UE is a mobile device, which subscribers use for voice calls and data transfer. [10 p. 8]

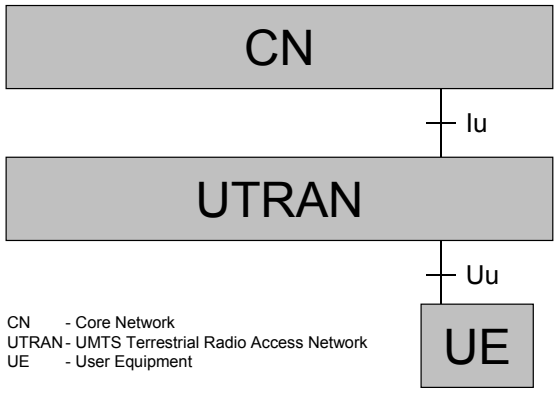

Figure 6: UMTS Architecture [11 p. 11]

3.1.1 Core Network

The UMTS Core Network (CN) is the platform for all communication services provided to the UMTS subscribers. The services include circuit-switched calls and routing of packet data. [10 p. 101]

3.1.2 User Equipment

User Equipment (UE) is the mobile device that subscribers use to access UMTS services, such as voice calls or data services. UE provides the end user side of the radio interface Uu, the other side is Node B in UTRAN. [10 p. 129]

3.1.3 UTRAN

Functions of UTRAN are listed in [11 p. 22-23]. They include:

- Transfer of user data.
- Functions related to overall system control.
- Radio channel ciphering and deciphering.
- Functions related to mobility (handover, paging support, positioning).
- Functions related to radio resource management and control.

The main task of UTRAN according to [10 p. 61] is to create and maintain Radio Access Bearers (RAB) for communication between UEs and Core Network. UTRAN (Figure 7) consists of one or multiple Radio Network Subsystems (RNS). Each RNS has at least one Radio Network Controller (RNC), which is responsible for controlling the use and the integrity of the radio resources. RNC controls one or multiple Node Bs (base stations), which are used to enable radio communication between RNS and UEs. [10 p. 70-71]

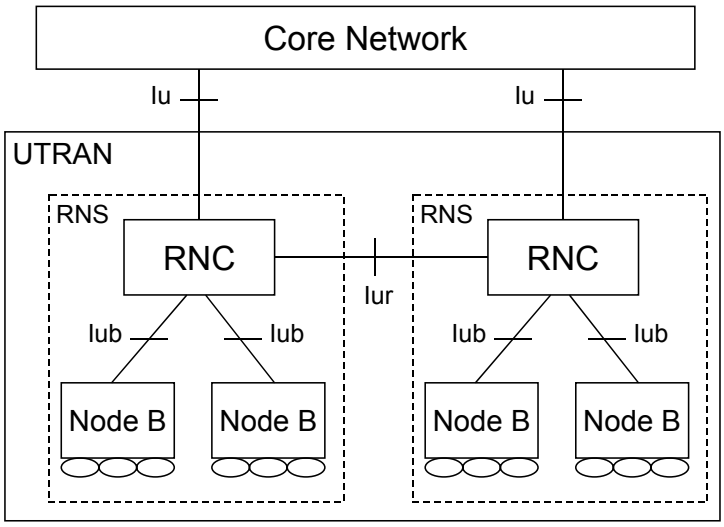

Figure 7: UTRAN architecture [11 p. 13]

3GPP has given names to interfaces between different network entities. The interfaces are Iu, Iur, Iub and Uu. Interface between Core Network and UTRAN is called Iu. Iub is the interface between RNC and Node B, Iur is the interface between two RNCs and Uu is the air interface between Node B and UE. [11 p. 7-8]

General protocol model for UTRAN used in these interfaces is described in Figure 8. The model is divided into horizontal layers and vertical planes, which are logically independent of each other. Independence makes it easier to make changes to one layer or plane without affecting the other layer or planes. For example, the transport network layer could be changed without major effect on the radio network layer. [11 p. 34]

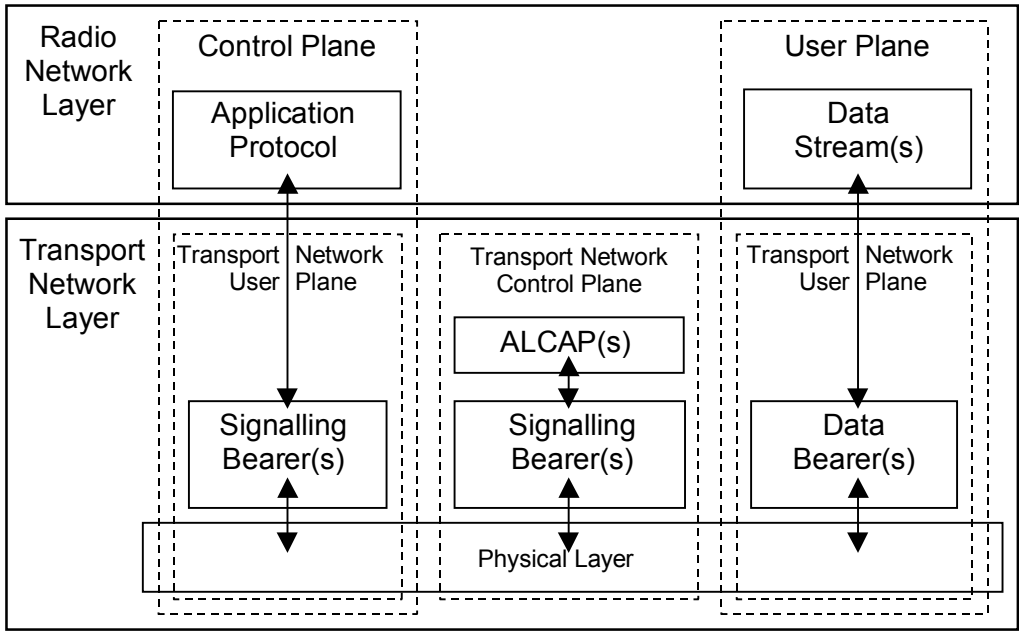

Figure 8: General protocol model for UTRAN interfaces [11 p. 34]

The protocol model consists of two layers. **Radio Network Layer** includes all protocols specific to UMTS system. Their common task is to control the management and use of radio access bearers across the UTRAN interfaces. **Transport Network Layer** contains protocols used for communication between network nodes. The protocols in this layer are not UMTS specific. [10 p. 211-212]

There are three planes in the protocol model. **Control Plane** includes Application Part (also called Application Protocol) and signalling bearers to carry Application Part messages. They are used for signalling, controlling and maintenance purposes. **User Plane** protocols are used for transporting user data. **Transport Network Control Plane** contains protocols needed to set up the transport bearers for the User Plane. [11 p. 34-35].

In **Iu** interface Radio Access Network Application Part (RANAP) provides signalling between UTRAN and CN. Functions of RANAP are listed in [12 p.17-18]. They include:

- Overall radio access bearer (RAB) management, which includes the RAB setup, maintenance and release.
- Management of Iu connections.
- Exchanging UE location information between RNC and CN.
- Paging requests from CN to the UE.
- Overload and general error situation handling.

Iur interface between two RNCs contains Radio Network Sublayer Application Part (RNSAP). Its functions are listed in [13 p. 25-26], they include:

- Management of radio links, physical links, and common transport channel resources.
- Paging.
- Serving RNC relocation.
- Measurements of common and dedicated resources.
- Reporting of general error situations.

Iub interface is more relevant to this thesis than Iu or Iur. Therefore it is presented in more detail in the next section.

3.2 Iub Interface

Node B and RNC communicate using Node B Application Part (NBAP) protocol over Iub interface. The complete protocol stack of Iub is shown in Figure 9.

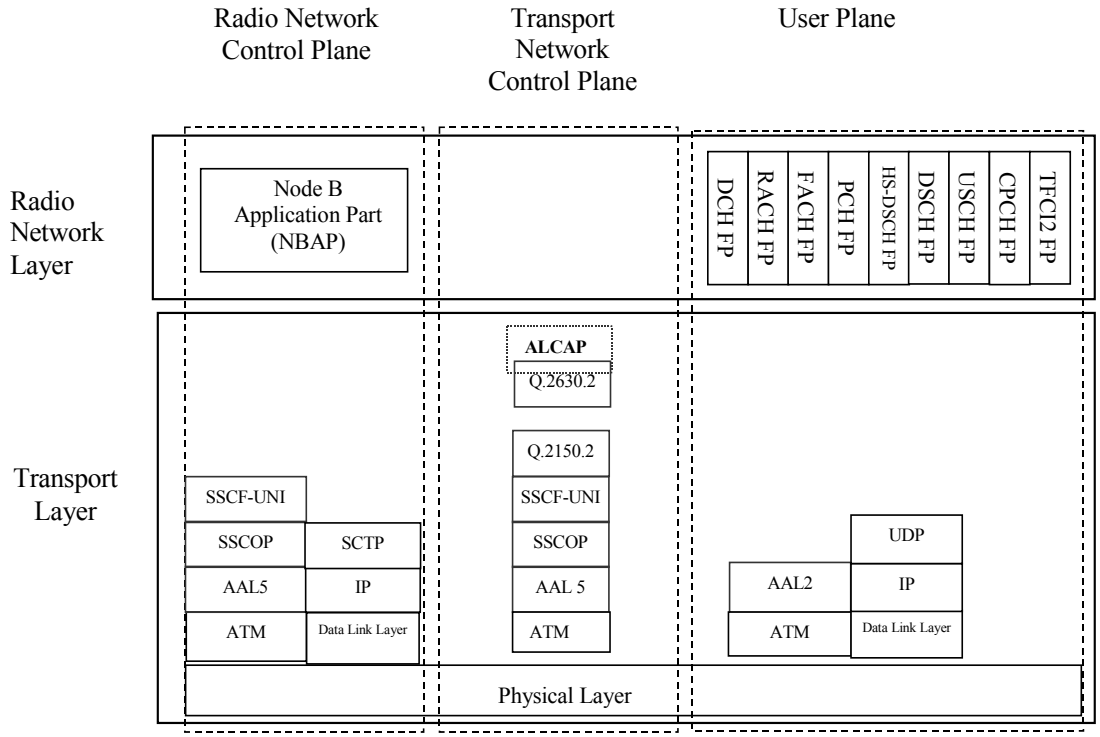

Figure 9: Iub interface protocol stack [14 p. 22.]

NBAP maintains control plane signaling, sets up dedicated user plane connections and controls resources in the Iub interface. Common resources refer to resources of Node B, which are not related to any particular UE. Dedicated resources refer to resources of Node B, which are reserved for one UE.

A complete list of functions of NBAP can be found from [14 p. 23-24]. Main functions are:

- Cell configuration management.
- Common transport channel management.
- Measurements on common resources.
- Radio link management.

- Measurements on dedicated resources.
- Reporting of General Error Situations.

Frame Protocols (FPs) in the radio network layer user plane are used to carry data over the common UTRAN transport network. They perform uplink and downlink data transfer, carry out control tasks, flow control, synchronization and timing. Transport layer deals with transport network technology chosen to transmit control and user plane data. [10 p. 251, 253]

3.3 WCDMA Channels

WCDMA system is designed to handle many different types of data traffic. To efficiently use bandwidth allocated for WCDMA, many types of channels are used. Here the term "channel" does not refer to radio frequency, but instead it describes allocated resources and related controlling functions. Channels are organized in three layers: logical, transport and physical channels. [10 p. 50]

Logical channels describe the type of information to be transmitted. *Transport channels* are defined by how and with what characteristics data is transferred over the air interface. They are divided into common and dedicated channels. *Physical channels* form the actual transmission media over air interface. They are defined by attributes such as carrier frequencies and codes. [10 p. 50] [15 p. 9]

Forward Access Channel (FACH) is a downlink channel carrying control information to UEs. One cell can contain several FACHs, but at least one of them is always available to all UEs. FACH can also carry packet traffic. *Paging Channel* (PCH) is used for paging UEs. Node B uses this channel when it wants to start a connection to a UE. Paging enables use of sleep-modes and power saving functions in user equipment. It is used in downlink direction only. *Random Access Channel* (RACH) is an uplink channel, which UEs use to transmit control information to the UTRAN, for example when a call is initiated. RACH can also be used to carry small amounts of uplink packet data. [15 p. 8] [16 p. 76]

3.4 Transport Network

One of the design aspects of the UTRAN is that it is as independent of the transmission protocol used as possible. There are two main alternatives for implementing transmission connections for UMTS: Asynchronous Transfer Mode (ATM) and Internet Protocol (IP). For the 3GPP release 1999 ATM was preferred because of more advanced quality of services than IP was offering at the time. In the future when IP's quality of service improves it can replace ATM in UTRAN network connections. This is the case in 3GPP release 4/5. [10 p. 219, 228]

3.4.1 ATM

ATM is an International Telecommunications Union (ITU) standard, which is used for transferring voice, video and data. ATM is a connection-oriented protocol, which uses small fixed size cells to transfer data. ATM technology combines benefits of both circuit switched and packet switched data transfer. Being a circuit switched protocol, ATM has a total control of the available transmission bandwidth and can guarantee high data rates and good quality of service. Cell relay and multiplexing of traffic enables efficient and flexible use of the bandwidth. Because ATM was developed for backbone networks it is assumed that the transmission medium is of good quality and the probability of transfer errors is small. ATM contains minimal error and flow control capabilities in order to reduce overhead and enable more efficient data transfer. [17 p. 66-71] [18 p. 328]

3.4.2 ATM Protocol Model

ATM protocol model consists of three planes and four layers. The three planes are user plane, control plane and management plane. The four layers are physical layer, ATM layer, AAL layer and higher layer. The protocol model can be described as a cube like in Figure 10. [19 p. 2]

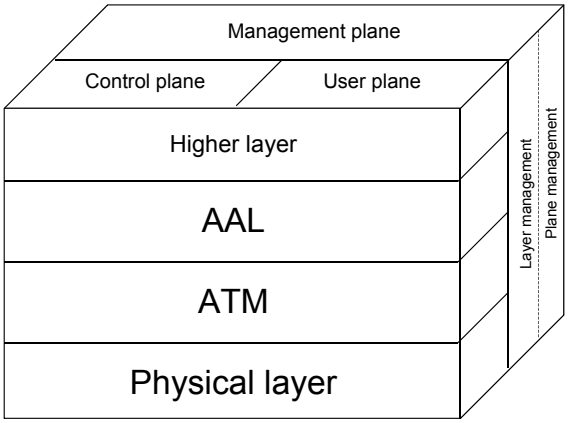

Figure 10: ATM protocol model [19 p. 2]

User plane deals with user information flow transfer, flow control and recovery from errors. *Control plane* handles call and connection control, and signaling necessary to set up, supervise and release calls and connections. *Management plane* has two parts. Plane management manages and coordinates functions related to the entire system, and provides coordination between all the planes. *Layer management* manages layer-specific functions such as the detection of failures and protocol problems. [19 p. 2]

Physical layer specifies transmission medium and signal encoding. Its purpose is to collect ATM cells from ATM layer and insert them into the selected physical transmission medium. *ATM layer* is common for all kinds of services. It provides packet transfer capabilities to higher layers by specifying use of logical connections and transfer of cells. [17 p. 125] [18 p. 328]

ATM cell has a fixed size of 53 octets. 5 octets are reserved for header and 48 for payload. Virtual Path Identifier (VPI) and Virtual Channel Identifier (VCI) specify the logical connection ATM cell belongs to. The relationship of transmission path, virtual channel and virtual path is shown in Figure 11. [17 p. 85] [20 p. 1]

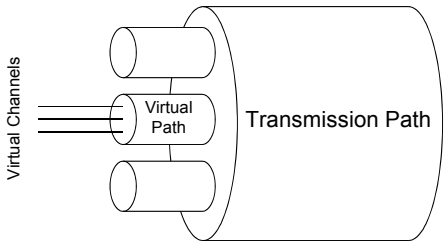

Figure 11: ATM connection relationships

ATM Adaptation Layer (AAL) is used to adapt different traffic types to use the same ATM layer. It maps data between higher layer protocols data units and ATM layer by fitting different types of data into 48 octet payload fields of the ATM cells. Functions of AAL include handling of transmission errors, flow control and timing control. Different AAL layers are needed to use ATM with various traffic types. Number of these layers has been limited by classifying service parameters (Figure 12), such as timing relation between source and destination, bit rate and connection mode. [21 p. 1]

	Class A	Class B	Class C	Class D
Timing relation between source and destination	Required		Not required	
Bit rate	Constant	Variable		
Connection mode	Connection-oriented			Connectionless

Figure 12: Service classification of AAL [21 p. 2]

[21 p. 2] gives some examples of applications where these classes might be used:

- Class A: Constant bit rate video.
- Class B: Variable bit rate video and audio.
- Class C: Connection-oriented data transfer.
- Class D: Connectionless data transfer.

AAL layers 1, 2, 3, 4 and 5 are specified in [22]. AAL1 corresponds to service class A, it provides real time, constant bit rate and connection-oriented data transfer. AAL2

corresponds to service class B, offering real time, variable bit rate service. AAL3 is designed for connection-oriented and AAL4 for connectionless data transfer. Layers 3 and 4 have been combined to layer 3/4. AAL5 offers the same services as layer 3/4. AAL2 and AAL5 are the most important adaptation layers for UMTS. In Iu, Iur and Iub interfaces AAL2 is used for user plane and AAL5 for control plane connections. Additionally AAL5 in used in packet switched user plane connections in Iu interface. [10 p. 232]

3.4.3 Applications in UTRAN

NBAP is transported over AAL5. NBAP is a signaling protocol, which requires assured in-sequence transfer capabilities from the signaling bearer [14 p. 23]. To use ATM as a signaling bearer it must be defined which ATM components are needed to provide necessary services for signaling. This is done in [23], which specifies components that are needed to use AAL for signaling purposes (SAAL). [24] defines signaling transport for NBAP (Figure 13), the model selected follows SAAL definition.

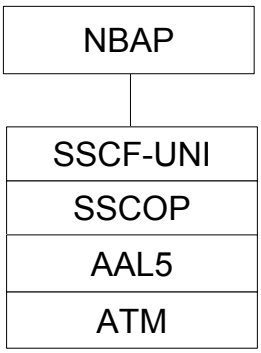

Figure 13: NBAP signalling over AAL5 [24 p. 6]

Service Specific Connection Oriented Protocol (SSCOP) is used with AAL to provide assured data delivery between AAL connection end points. Complete set of features is listed in [25]. SSCF-UNI layer [26] is needed to map services between NBAP and SSCOP.

Figure 14 describes how common transport channels RACH, FACH and PCH are transported over AAL2. Service Specific Segmentation and Reassembly sublayer (SSSAR) is used for segmentation and reassembly of AAL2 data. It provides bandwidth-efficient transmission of low-rate, short, and variable length packets in delay sensitive applications. [27 p. 8] [28 p. 1]

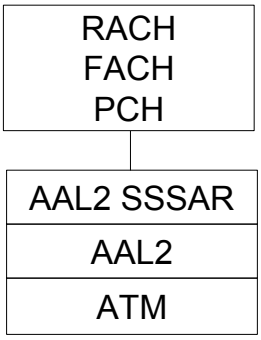

Figure 14: Common transport channels over AAL2 [27 p. 8]

4 Node B

In UMTS a Node B is located between Iub and Uu interfaces (Figure 15). Its main task is to establish the physical implementations of the Uu interface towards UEs and Iub interface towards RNC. With Iub this means implementation of ATM transport and protocols described in Figure 9. With Uu this means implementation of WCDMA radio access physical channels and transforming information from transport channels to the physical channels according to RNCs requirements. [10 p. 63]

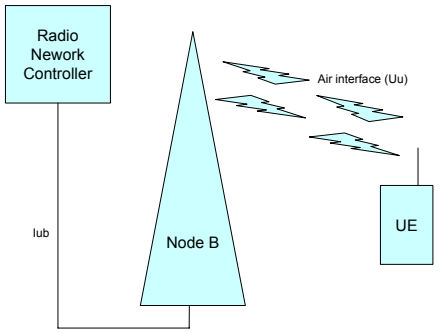

Figure 15: Node B interfaces Iub and Uu

4.1 Structure

The internal structure of a Node B is vendor-specific, but this could change in the future if standardized architecture gains popularity. Logical structure (Figure 16) from the network point of view is generic. Iub side of a Node B can be seen consisting of a common transport entity and Traffic Termination Points (TTP). Common transport represents transport channels, which are common for all UEs in the cell and those used for initial access. Common transport entity also contains one Node B Control port used for operation and maintenance (O&M) purposes. A TTP consists of a number of Node B Communication Contexts, which in turn consist of all dedicated resources required by the UE, when it is in dedicated mode. [10 p. 63, 64]

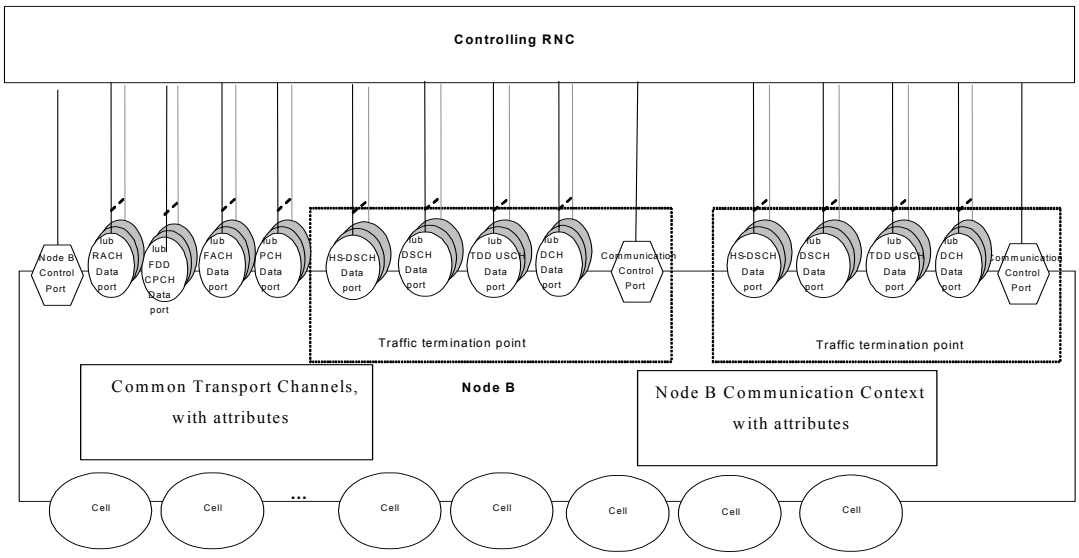

Figure 16: Node B logical structure [29 p. 15]

Node B contains cells, which are the smallest radio network entities. A cell has an own scrambling code and identification number (cell ID), which UE uses to differentiate between cells. One cell may have several Transmitter-Receivers (TRXs, also called carriers). TRX delivers broadcast information towards UEs. The TRX is physically a part of the Node B performing various functions, data flows are converted from Iub connection to the radio path and vice versa. [10 p. 64, 65]

4.2 Hardware and software

This section presents the most important Node B hardware and software entities concerning this thesis. Figure 17 shows simplified hardware architecture of a Node B. It describes how the units are located between Node B's Iub and Uu interfaces. In reality the units might not be directly connected to each other and there could be other units between them.

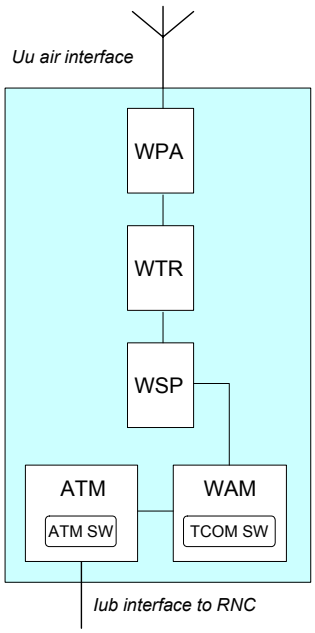

Figure 17: Node B units and software

ATM software (ATM SW) is located in the ATM unit. It provides ATM connections such as AAL 5 NBAP signaling links and AAL2 user plane links. The interface between TCOM SW and ATM software is used by TCOM SW to setup, modify and release connections. [30 p. 42] [33 p. 5]

Wideband Application Manager (WAM) unit contains TCOM SW and also has O&M functionality [31 p. 68]. **Telecom Software** (TCOM SW) communicates with RNC using NBAP protocol. After receiving an NBAP message TCOM SW performs required actions to fulfill RNC's request. These actions include reserving, configuring and releasing resources of the Node B. For this purpose the TCOM SW communicates with different subsystems of the Node B, such as ATM software, WPA and WTR units via specific communication interfaces. Responsibilities of TCOM SW according to [30 p. 31] include:

- Part of base station start up
- Common channel handling (setup, release, reconfiguration)
    - RACH
    - FACH
    - PCH

- Dedicated channel handling (setup, release, reconfiguration)
    - Radio links
- Recovery and error handling
- Test support

Wideband Signal Processor (WSP) unit performs channel processing, coding and decoding functions. **Wideband Transmitter and Receiver** (WTR) unit has two functions. On transmit side it receives digital data from summing unit and modulates and performs up conversion of the transmitted carrier. On receive side the unit performs channel selection and down conversion for the selected carrier. The signal is digitized and transmitted to summing unit. The interface between TCOM SW and WTR unit is used by TCOM SW to initialize the WTR and configure its channels. WTR sends power level reports to TCOM SW. [30 p. 39] [31 p. 69]

Wideband Power Amplifier (WPA) unit amplifies the signal from WTR to be transmitted via antennas of the base station. Signal amplification is mostly done by analogue amplifier hardware, but as part of the WCDMA system the unit offers control for initialization, activation, deactivation and monitoring. The interface between TCOM SW and WPA unit is used by TCOM SW to initialize the WPA and configure its channels. WPA sends power level reports to TCOM SW. [30 p. 40] [32 p. 13]

4.3 Open base station architecture initiative

Developing a base station from scratch requires a lot of effort and is very costly. When each manufacturer develops an own proprietary solution, effort is multiplied. Vendors of base stations, modules and components have formed an organization called The Open Base Station Architecture Initiative (OBSAI) to create a set of specifications for open base station architecture. Open architecture reduces development cost and effort, since manufacturers don't need to develop everything themselves. More R&D resources are available for innovating new features and applications. Product development cycle is shortened and new products will be available to end-users more often and with cheaper price. [34 p. 4]

OBSAI architecture is shown in Figure 18. It defines four functional modules: transport, baseband, radio frequency (RF) and control. The standardized interfaces between the modules are called reference points (RPs).

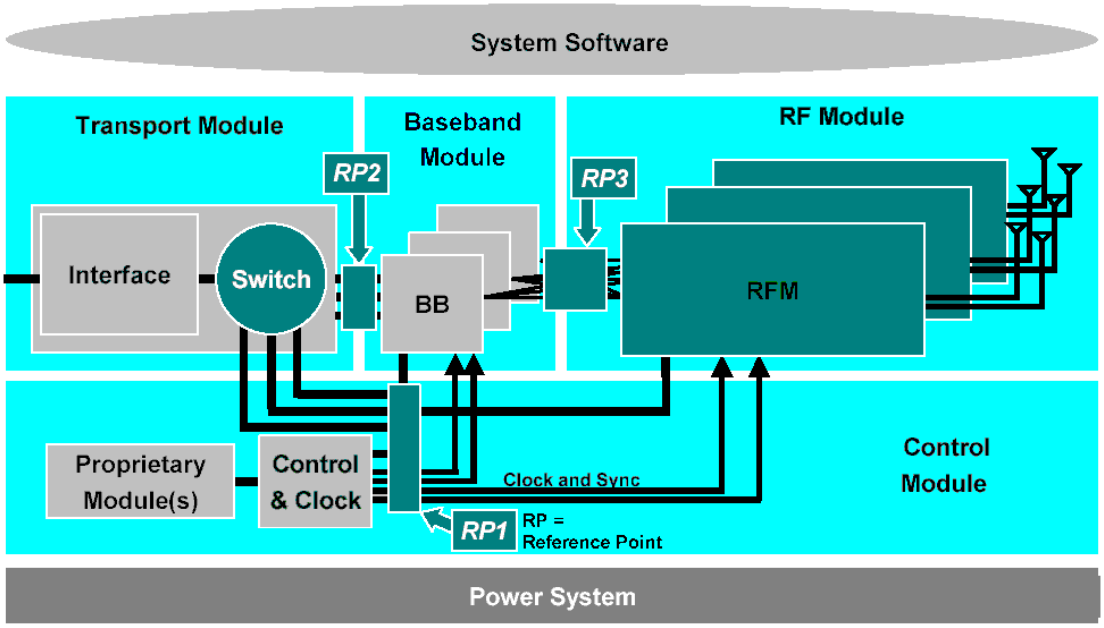

Figure 18: Obsai BTS architecture [34 p. 10]

Module functionality according to [34 p. 9] is:

- **Transport module** provides adaptation between external network and standard base station internal interfaces.
- **Baseband module** (Processing module) provides a set of channel modems and the baseband processing for the air interface.
- **RF module** (Radio module) provides RF transceivers, RF amplification and conversion between digital baseband signals and analogue RF signals. The RF module can be located remotely or within the base station.
- **Control module** provides the primary control processing for the base station. The control module supervises all base station activities, monitors the base station and reports its status and performance to a Base Station Manager or Element Management System.

5 TTCN and TTCN tester

Tree and Tabular Combined Notation (TTCN) is a test notation, which is used in this thesis for test case implementation. TTCN tester is used to execute the TTCN test cases against the Node B.

5.1 TTCN specification

ISO/IEC 9646 standard "Framework and Methodology of Conformance Testing of Implementations of OSI and CCITT Protocols" introduces the concept of abstract test suites containing abstract test cases. According to the framework, tests should be described using black box model, using available interfaces. Test suites and test cases are described in a formal language called TTCN, which is specified in [35].

TTCN is independent of test methods, layers and protocols. With help of TTCN, abstract test cases can be expressed in standardized test suites. TTCN is used especially when testing telecommunications protocols, but it can be used to test almost anything if the test can be performed using a black box model. TTCN enables use of Abstract Syntax Notation 1 (ASN.1), which is used in definition of 3GPP protocols. For example NBAP ASN.1 definitions can be found from [14 p. 365-666]. Manual labor and probability of human errors are reduced when definitions for protocol messages can be directly imported from specifications. [36]

5.1.1 Test suite, test cases and test steps

In TTCN, all the tests cases are contained in a test suite, which is stored into a file. The file can be either in a human readable graphical form (TTCN-GR), or in a machine processable form (TTCN-MP), which suitable for transferring descriptions between machines and for automated processing. In practice the test suite is stored in a machine processable form, and modifications are made with some application having graphical user interface. In addition to the test cases, TTCN test suite contains all related information that is needed in test cases, such as test steps, constants, variables, protocol data units (PDUs) and timers. Test suite is divided into five parts: suite overview, import part, declarations part, constraints part and dynamic part. [35 p. 9-10]

Test suite overview provides information needed for general presentation and understanding of the test suite. It contains complete lists of test cases and test steps in the test suite. Import part declares objects that are imported from a source object, for example from an ASN.1 file. Importing is equivalent of having a copy of the imported object in the test suite. Declarations part defines and declares all the objects used in the test suite. These include test suite data types, operations, parameters, constants, variables, timers, Abstract Service Primitive (ASP) types and PDU types. Constraint part specifies values of the ASP parameters and PDU fields that are to be sent or received by the test system. Dynamic part is the main part of the test suite. It contains behavior descriptions of test cases, test steps and defaults. [35]

Test suite is hierarchically structured in three parts; test groups, test cases and test steps. Test groups are used to arrange test cases in logical way, so that proper test cases can be found easily. Division into different groups can be based on, for example, tested functionality or procedures, software or hardware version, test configuration, etc. Grouping is not needed if there are only few test cases. [35 p. 15]

Test case contains description of anticipated events and verdicts assigned to each sequence. Test case can be described as a tree structure, in which nodes represent TTCN statements, and leaves represent verdict assignments (Figure 19). [35 p. 92]

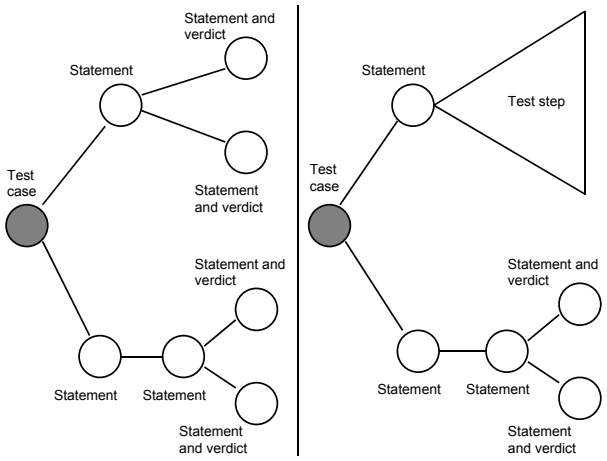

Figure 19: Test case behaviour structure [35 p. 94]

Test case tree can be divided into subparts, called test steps. They are used to hide complexity in large test cases. When TTCN code is edited, it is easier to understand what happens in a test case if it is divided into test steps. Test steps should be given names, which describe the test step functionality well. Test steps are used to reduce redundancy in test cases. Often needed functionality can be written into a test step, which is then called by all the test cases that need such functionality. Test steps can be parameterised, which increases their reusability. [35 p. 15, 95]

5.1.2 Concurrency in TTCN

In testing, there is often need to exchange messages in multiple interfaces of the system under test (SUT) at the same time. This is possible with TTCN, because it enables executing test components concurrently (Figure 20). In concurrent testing the components involved are called Main Test Component (MTC) and Parallel Test Components (PTCs). There is always one MTC, which controls the execution of the test. There can be zero or more PTCs, usually at least as many as there are interfaces to be tested in the SUT. Test components communicate with the SUT by using Points of Control and Observation (PCOs). Test components communicate with each other by using Coordination Messages (CMs), which are delivered through Coordination Points (CPs). PCOs can be though of consisting of two First In First Out (FIFO) queues; one for sending abstract service primitives (ASPs) and protocol data units (PDUs) to the SUT, the other for receiving ASPs and PDUs from the SUT. Similarly, CPs can also be described as being two FIFO queues. [35 p. 13]

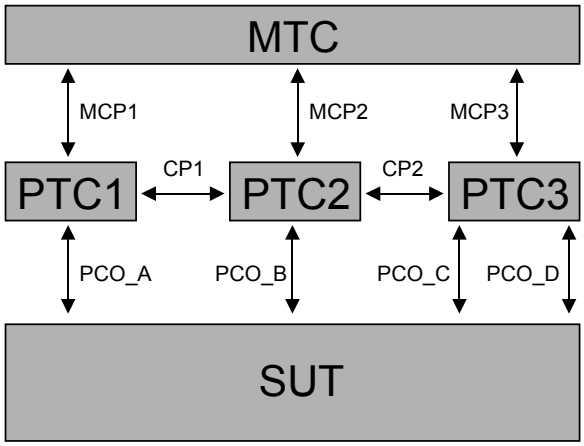

Figure 20: Test component configuration [35 p. 14]

5.1.3 Other TTCN features

Constraints are used to match information received from the SUT. For example, in a test case it could be expected, that after sending a certain message to the SUT, it should reply with a message containing series of certain values. In the receiving part of the test case the expected values can be written as constrains. If the values received match the constraints, that part of the test case can be considered successful. If the values differ, it can be assumed that the SUT has not behaved as it is expected, and this part of the test cases can be considered as failed. A constraint can be made flexible, for example, it can be written so that some value is expected, but the actual value can be anything. These kinds of features ease testing in situations, where there is not precise knowledge of possible response values, or when the values are not important as far as the test case is concerned. Parameterization of a constraint is also possible, which increases their usability. [35 p. 73]

When PDUs need to be sent or received from a message interface, there could be additional information (headers, padding etc.) added around PDU. Constraints can be used as containers for PDUs to make handling of such additional information easy. For example, when a PDU is sent to an interface, which requires routing information to be added, the PDU could be fed to a sender constraint. That constraint would add required routing information to the PDU and the constraint could be sent to the interface.

Constraints used in this way “wrap” information around PDUs, in this thesis these kinds of constraints are called “wrapper constraints” or “wrappers”.

Timers are often needed, especially with communications protocols. Timers are used to set time limits to events. For example, if the SUT is expected to send a certain message within X seconds, a timer can be used when verifying the correct behavior. Timers can also be used to cause delays, which can be needed for example between executing different test steps. [35 p. 50]

TTCN verdicts are used to judge if a test case has been executed successfully or not. In TTCN there are three verdicts: PASS, FAIL and INCONCLUSIVE. In a test step there can be a preliminary verdict indicating the success or failure of that test step. The knowledge of the preliminary verdict is stored into global variable R. When there are multiple test steps, each having a preliminary verdict, a final verdict is formed by changing the variable R according to Table 4:

Current value of R	**Entry in verdict column**		
	(PASS)	**(INCONC)**	**(FAIL)**
NONE	PASS	INCONC	FAIL
PASS	PASS	INCONC	FAIL
INCONC	INCONC	INCONC	FAIL
FAIL	FAIL	FAIL	FAIL

Table 4: Calculation of the variable R [35 p. 121]

As can be seen from the table, the verdict INCONCLUSIVE overrides the verdict PASS, and the verdict FAIL overrides all other verdicts. This ensures that incorrect behavior will be noticed when test cases are executed. Correct updating of the variable R is done automatically by TTCN.

5.1.4 TTCN-3

Tabular structure on test specifications used in versions 1 and 2 is sufficient for protocol testing, but it is quite inflexible and the test notation lacks readability. European Telecommunications Standards Institute (ETSI) has specified a new version of TTCN

(TTCN-3) in ES 201 873 series. TTCN-3 has look and feel of a modern programming language. [37]

According to [38 p. 15] TTCN-3 is "applicable to the specification of all types of reactive system tests over a variety of communication interfaces. TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing." The language has been renamed to better reflect its characteristics. TTCN now stands for Testing and Test Control Notation. [38 p. 15]

5.2 Nokia's Node B TTCN Tester

Node B TTCN Tester is a tool for testing Node B. Originally the focus of the tester was on testing NBAP signaling, Node B internal interfaces and Uu interface. Later the tester was extended to support testing of O&M Iub, focusing on O&M Iub procedures and alarm handling. The following list shows how the tester is used in various phases of Node B testing. System Under Test (SUT) and used interfaces are presented. [3 p. 10-13]

- Module testing
    - SUT: WAM and ATM sub unit.
    - Interfaces: Iub, simulated Uu
- System testing
    - SUT: Node B
    - Interfaces: Iub, Uu
- O&M testing
    - SUT: Node B
    - Interfaces: Iub, Uu, Node B internal interfaces.
- System component testing
    - SUT: WAM (TCOM SW)
    - Interfaces: Simulated Iub, Node B internal interfaces.

5.2.1 General architecture

The tester system has been divided into two subsystems. Executable Test Suite Execution Unit (ETSEU) is responsible for execution of TTCN test cases and providing a user interface to test engineer. Protocol Execution Unit (PEU) is responsible for communicating with the Node B, which means it must have the hardware for physical connections and it must implement required protocols and message interfaces. Figure 21 describes the tester platform for system component testing in more detail. Hardware interface to the Node B consists of an Ethernet card, which is connected to WAM unit of the Node B. In system and O&M testing configuration also an ATM card would be present to provide Iub interface.

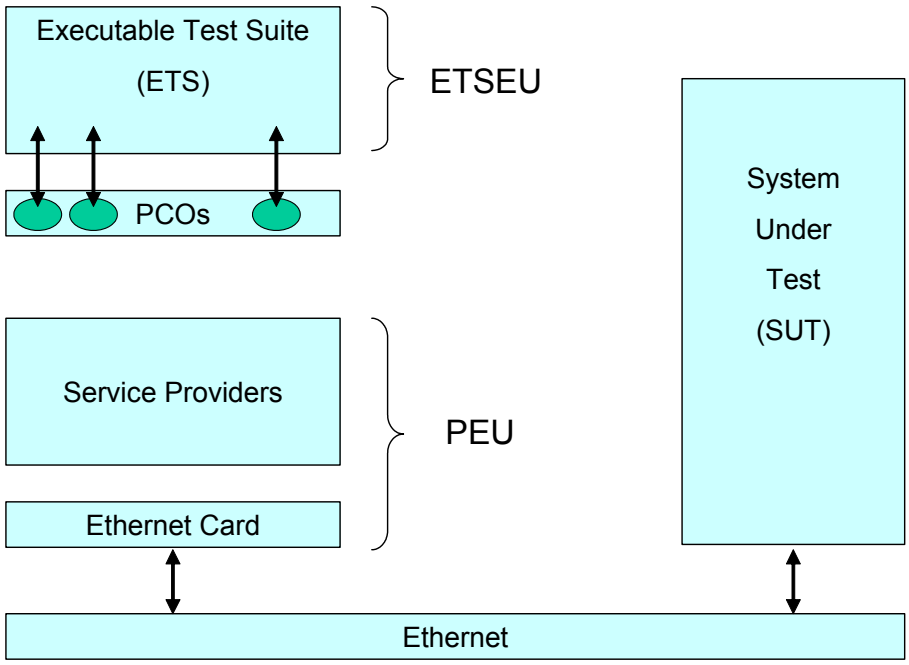

Figure 21: Generic tester platform [39 p. 21]

The ETSEU consists of the Executable Test Suite (ETS) and Points of Control and Observation (PCOs). The ETSEU contains test cases and runs MTC and PTCs. The test cases are created with a TTCN editor, such as ITEX by Telelogic, and they are compiled to executable binaries with Nokia internal tools.

The PEU is a hardware unit executing the service providers. Service providers are the different protocol layers needed to communicate with the Node B. They are accessible to the ETSEU via PCOs. PCOs are divided logically so, that each of them provides communication port to an independent interface. In the complete tester there are PCOs for NBAP, O&M, UE, WAM, WSP, WPA, WTR and ATM to name a few. In testing only a subset of them is needed depending on the test configuration. PCOs use lower layer service providers by sending and receiving Abstract Service Primitives (ASPs) through their Service Access Points (SAPs). ASPs could contain for example Protocol Data Units (PDUs), which in turn carry Information Elements (IEs). In addition to communication PCOs, some PCOs are used for configuring the service providing protocol layers. [39 p. 22]

5.2.2 From TTCN editor to executable binary

First test case writer creates test cases with ITEX TTCN editor and saves them into TTCN machine processable mp files. Multiple mp files are created because test suite is modular and functionality is divided into smaller parts. Text filter tool converts mp files into an MP-format understood by the TTCN compiler front end. Conversion is needed because the ITEX editor uses proprietary extensions to the TTCN standard. Each TTCN module is compiled into its own binary file with the extension *.tm. When all modules have been compiled, the TTCN compiler front end is used to build the Abstract Test Suite (ATS), which is then converted into C source and configuration files. Other C language source files are either created manually like the user mapping functions and PCO & Coordination Point (CP) definitions, or automatically using the Compiler for ASN.1 (CASN) for ASN.1 mapping functions. Gcc compiler is used to compile all the C modules on the ETS. The process is shown in Figure 22. [39 p. 54-55]

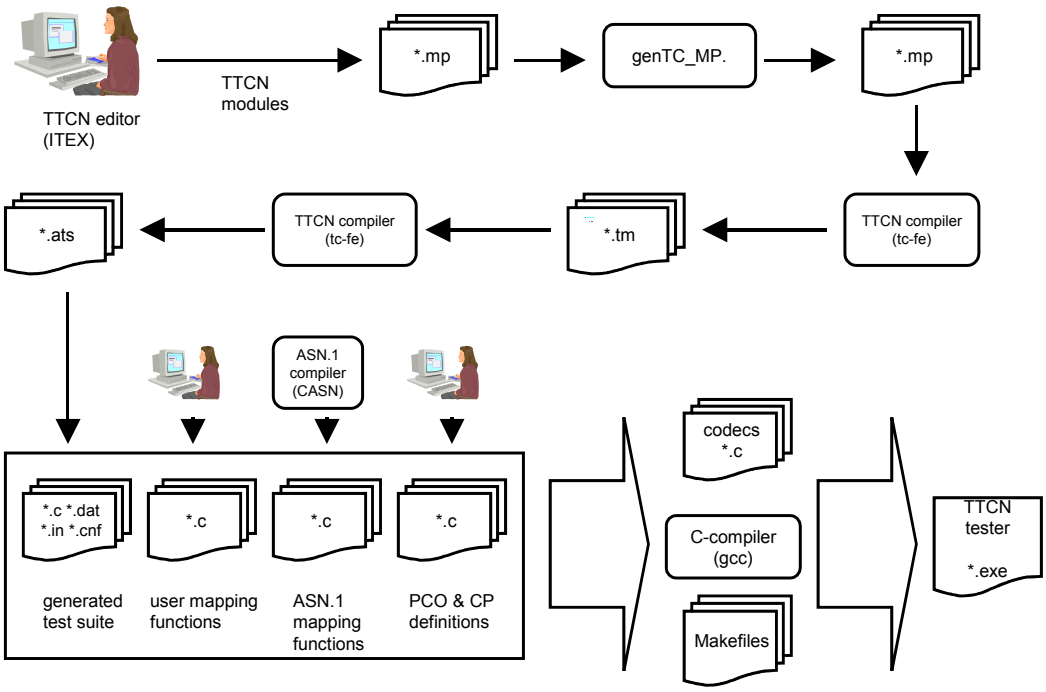

Figure 22: Tool chain [39 p. 54]

5.2.3 Using the tester

The tester can be started after the executable test suite has been successfully built. Test engineer first starts the PEU to enable test suite communication with the Node B. Next the ETSEU can be started. When the tester starts windows are created for the MTC and all the PTCs, which have been built. An example view of the tester is shown in Figure 23. A test case called *CellSetup_1_of_3_WPA_WTR* has been started in the MTC. The test case has started a test step called *s_iub_CellSetup_1_of_3_MT* in the Iub_PTC_1 and *s_wpa_wtr_CellSetup_MasterWAM_MT* in the WPAWTR_PTC_1. In this case no test step has been started in the WAM_ATM_PTC_1. When the test case has been executed, final verdict PASS or FAIL is displayed in the MTC.

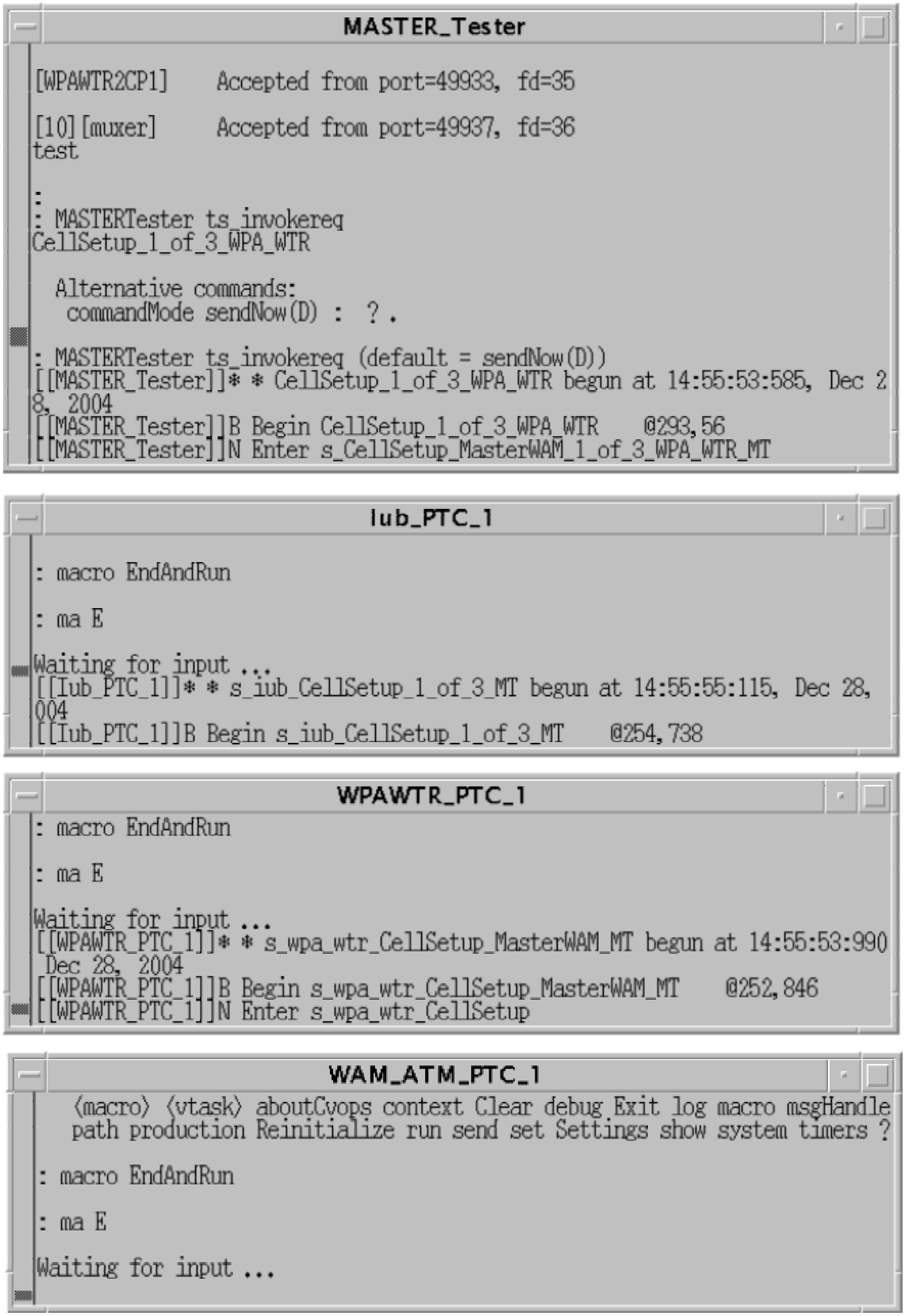

Figure 23: Tester running

5.2.4 TASSU interface

When a unit in Node B is component tested, the test system needs to communicate with Node B internal message interfaces. The system must be able to receive messages from and send messages to units, which are not normally communicating with Node B external interfaces such as Iub or Uu. Somehow those units and their message interfaces must be made visible outside the Node B. A special tool has been developed for this purpose called Testing Analyser and Simulator for Systems and Units (TASSU). TASSU is a stand-alone application, which test engineer uses to monitor, send or receive messages from Node B internal message interfaces. TASSU application communicates with TASSU Router (TR) and TASSU Test Support (TTS), which are running in the WAM board of the Node B. TR is responsible for receiving messages from TASSU and delivering them to correct units inside Node B. It also sends messages from the units to TASSU. TTS is used for message monitoring between Node B units. [40 p. 12]

TASSU interface has been implemented in the TTCN tester, which gives the tester powerful means of communicating with the Node B units. The tester uses TCP/IP connection over the Ethernet to connect to the TR on the WAM board. For TTCN testing TASSU interface offers two possibilities, to monitor the internal messages, or re-route them to the tester. [41 p. 16]

Message monitoring and re-routing are based on so called API header. Each message contains an API header, which has fields for Sender and Receiver information. Address of sending unit is stored into Sender field and address of unit that should receive the message is stored into Receiver field. Both TR and TTS check Receiver field of messages. [40 p. 16, 41]

Figure 24 describes monitoring with TTS. When messages are monitored, TTS sends a copy of the monitored message to the tester's TASSU interface. The tester can configure TTS so that messages containing certain Receiver values in the API header are delivered to the tester. In the test cases monitoring is used when it is checked that certain messages are exchanged between Node B units. For example, if test case

specification says that event X must cause message Y to be exchanged between units A and B, it would be implemented by configuring TTS to monitor the interface between units A and B, and then causing event X to take place. The tester would first configure TTS and set monitoring parameters, second it would trigger event X, and finally it would check for message Y. Monitoring is useful, when messaging between two Node B units must be observed, but not disturbed. The tester gets only a copy of the original message, meaning that the receiver unit will receive the message in normal way. In Figure 24 message sent by unit A is received by unit B, which will reply. If the tester would also reply, unit A would receive two replies, which would disturb test execution. In case where the tester must take an active role and simulate Unit B it is better to use re-routing of messages. [40 p. 39]

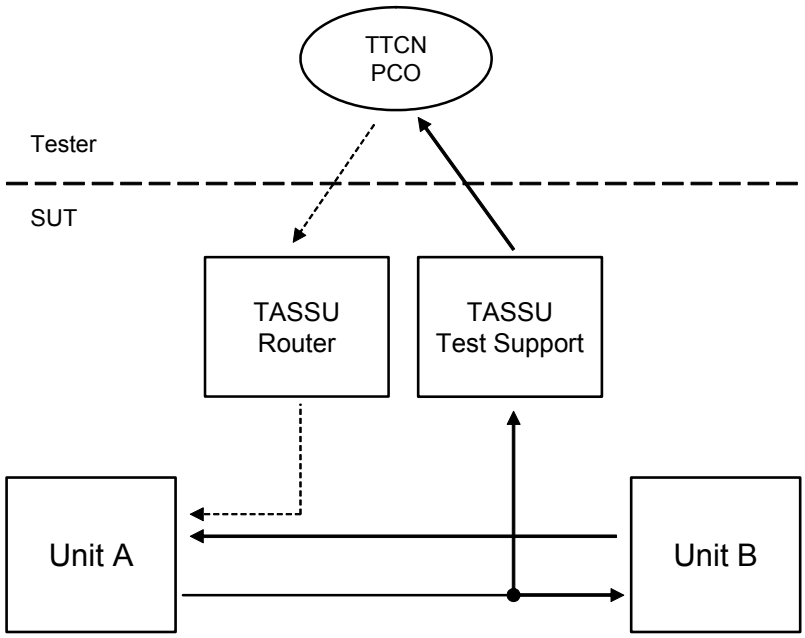

Figure 24: Monitoring with TASSU Test Support [41 p. 20]

Re-routing with TASSU Router is described in Figure 25. Re-routing method is used when messages meant to some unit need to be re-directed to the tester instead of the original unit. This is the case for example in system component testing of TCOM SW in the WAM unit. When Node B is in operation, WAM unit communicates with several other units inside the Node B. In system component testing not all units are present, or

have required capability or compatible software to run tests with WAM. The TTCN tester can be used to take role of the missing units, and WAM can be fooled to run without all required hardware. In this situation the tester configures TR to re-route messages, which have the address of the missing unit in the Receiver field of API header. Those messages are then received in the tester instead of target units. Re-routing differs from monitoring so, that the unit to which WAM is sending messages will never receive them. This ensures that the WAM will not receive duplicate replies, one from the tester and the other from the real hardware unit. In Figure 25 unit A sends a message to unit B. The tester has configured TR to re-direct the message to the tester. As unit B will never receive the message from unit A, it will not send a reply. Thus the tester can simulate unit B by sending its own reply message to unit A. [40 p. 16]

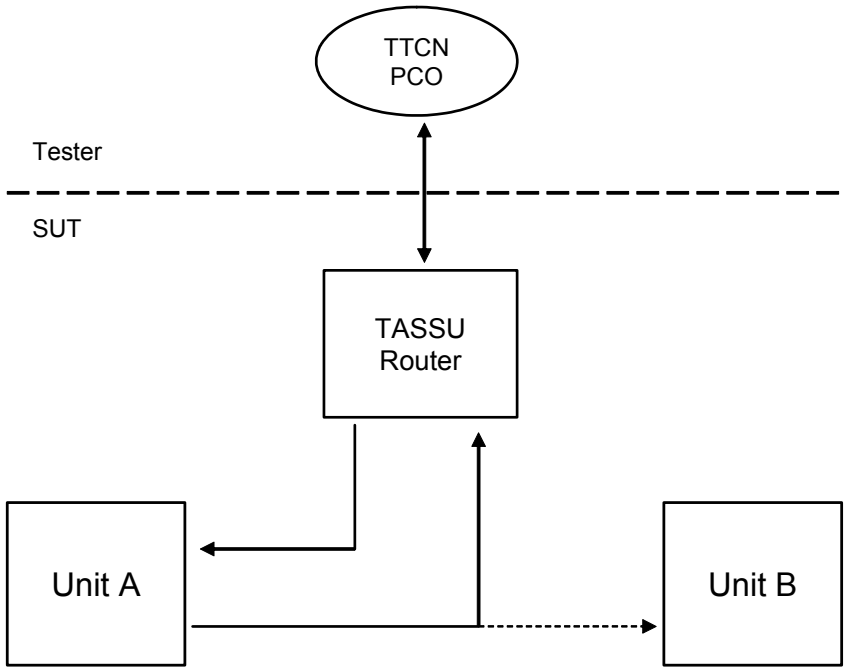

Figure 25: Re-routing with TASSU Router [41 p. 19]

5.2.5 Test environment for system component testing

Test scenario for system component testing is shown in Figure 26. The tester has a TCP/IP Ethernet connection to TR located on WAM unit of Node B. The SUT is TCOM SW located in the WAM unit.

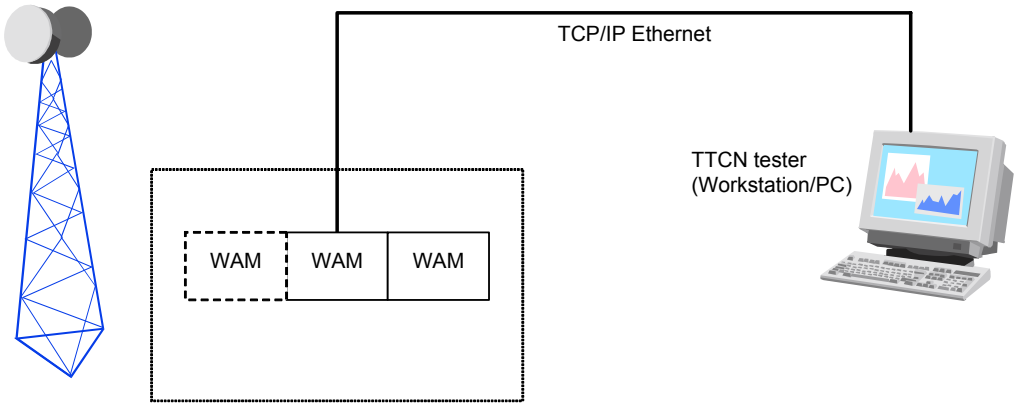

Figure 26: Test scenario [39 p. 19]

SUT environment for system component testing is shown in Figure 27. The tester simulates RNC, ATM SW, WSP, WTR and WPA by using TR and TTS to exchange messages over the physical interfaces. [39 p. 18]

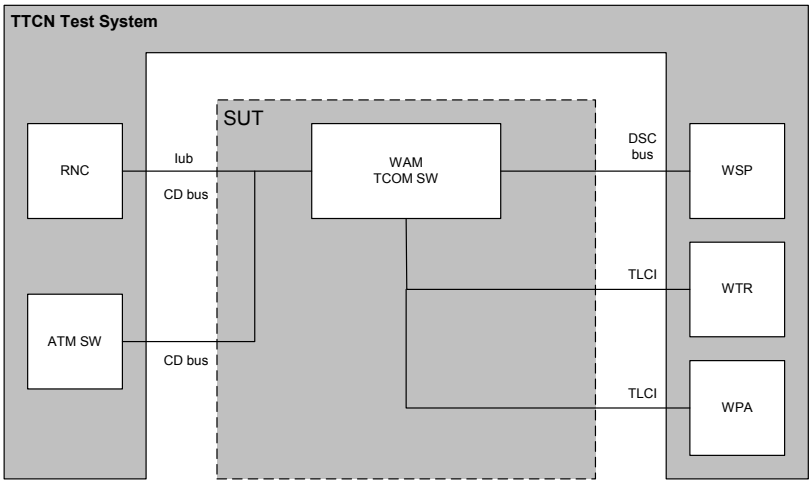

Figure 27: SUT environment for system component testing [39 p. 20]

Simulation of RNC is done by sending and receiving NBAP PDUs. Normally ATM Iub connection is used for NBAP signaling between RNC and Node B, but it cannot be used in system component testing configuration, because Node B's ATM SW is out of operation. The tester uses TTS to monitor NBAP PDUs in the CD-bus, which shared memory bus that TCOM SW and ATM SW use to communicate with each other. NBAP PDUs from the tester to TCOM SW are sent using TR. In this thesis this interface is called TASSU Iub. *Simulation of ATM SW* is done like simulation of RNC. TTS is used for monitoring messages from TCOM SW to ATM SW in the CD-bus. TR is used for sending messages from the tester to TCOM SW. TCOM SW and WSP communicate via a DSC-bus (Data, Signaling and Control). *Simulation of WSP* is done by configuring TR to re-route TCOM-WSP messages in the DSC-bus to the tester. TR is used when the tester is sending WSP messages to TCOM SW. WPA and WTR communicate with TCOM SW via a TLCI bus. *Simulation of WPA and WTR* is done by configuring TR to re-route TCOM-WPA and TCOM-WTR messages in the TLCI-bus to the tester. TR is used when the tester is sending WPA and WTR messages to TCOM SW. In Figure 28 it is shown what kind of messages are exchanged between TCOM SW and the entities simulated by the tester (RNC, ATM SW, WSP, WTR and WPA). [33 p. 60] [33 p. 67] [40 p. 25]

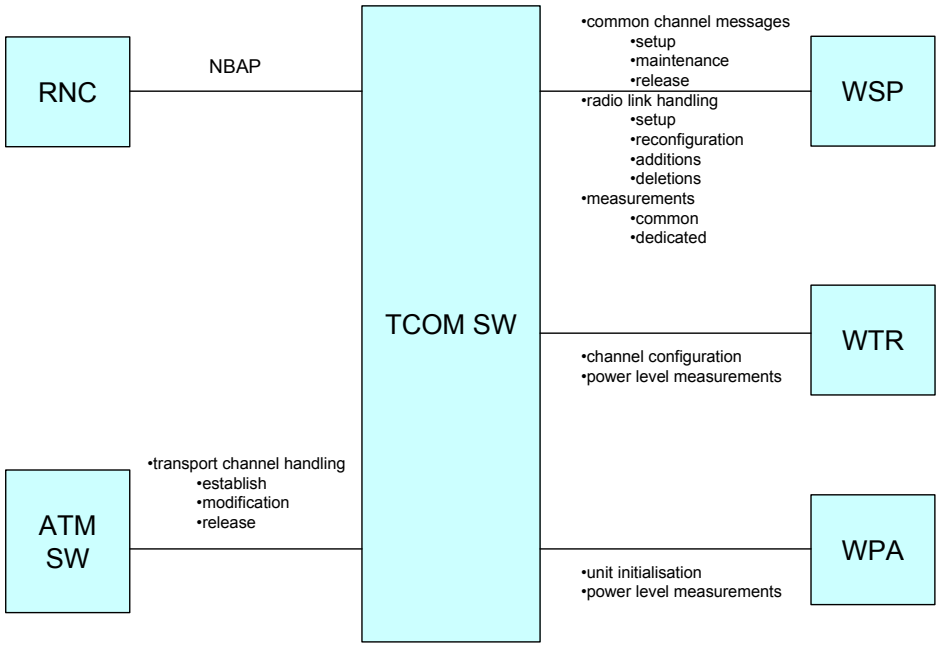

Figure 28: Interface messages [32 p. 32]

5.2.6 TTCN architecture for system component testing

TTCN architecture for system component testing is shown in Figure 29. Master Test Component (MTC) connects to parallel test components (PTCs) for Iub, WSP, WPA, WTR and ATM interfaces via coordination points (CP). It is notable that the same PTC is used for WPA and WTR (WPA_WTR_PTC). Each PTC connects to WSP service provider through points of control and observation (PCO). WSP service provider is connected to the SUT using TASSU connection.

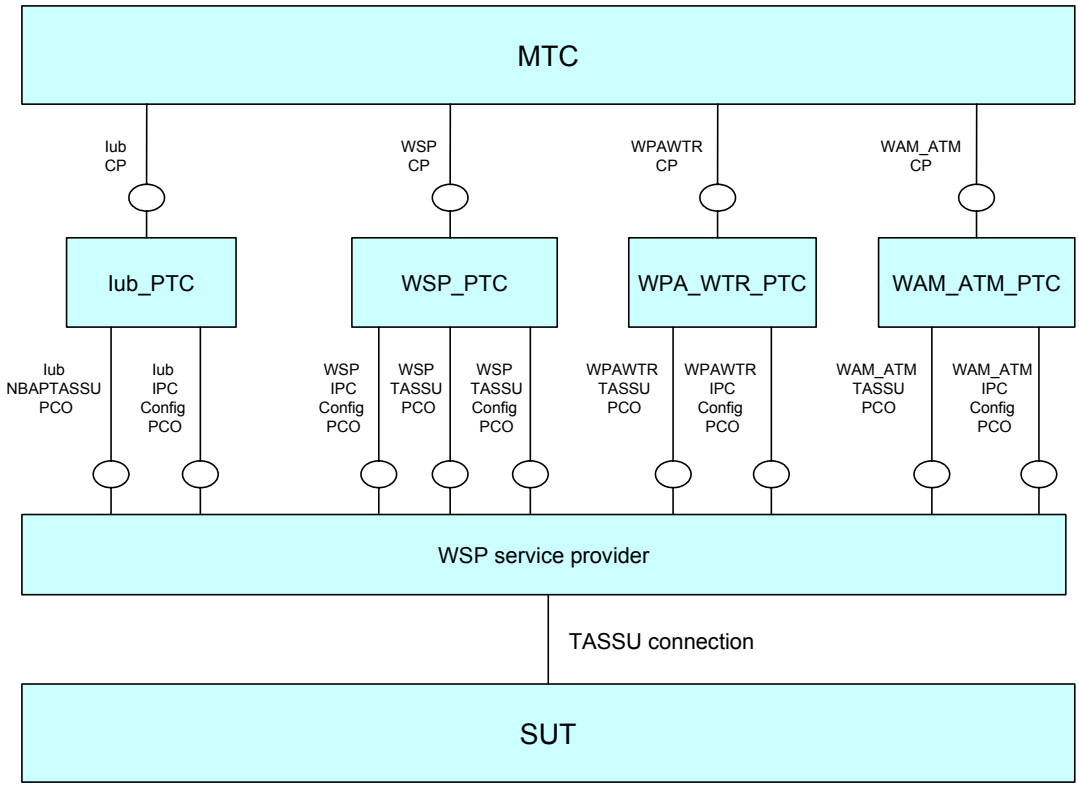

Figure 29: TTCN architecture for system component testing [39 p. 53]

Communication between the tester and the SUT is divided logically between the PTCs so that each PTC is dedicated for a specific interface. There are more than one PCO per PTC, because one is used for exchanging interface messages and the other is needed for establishing, modifying and releasing the PTC connection between the ETSEU and the WSP service provider. The purpose of the PTCs and their PCOs is described in Table 5.

PTC	PCO	Description
Iub_PTC		
	Iub NBAPTASSU PCO	The tester simulates RNC through this PTC. Used to transfer Common and Dedicated NBAP over TASSU connection.
	Iub IPC Config PCO	For creating connection between the testing PCO (Iub NBAPTASSU PCO) and the PEU.
WSP_PTC		
	WSP TASSU PCO	The tester simulates DSP units through this PTC.
	WSP IPC Config PCO	For configuring the connection between the WSP_PTC in the ETSEU and the WSP service provider in the PEU.
	WSP TASSU Config PCO	Used for establishing and deleting TASSU connections between the tester and the SUT.
WPA_WTR_PTC		
	WPAWTR TASSU PCO	The tester simulates WPA and WTR units through this PTC.
	WPAWTR IPC Config PCO	For configuring the connection between the WPAWTR_PTC in the ETSEU and the WSP service provider in the PEU.
ATM_PTC		
	WAM_ATM TASSU PCO	The tester simulates ATM SW through this PTC.
	WAM_ATM IPC Config PCO	For configuring the connection between the WAM_ATM_PTC in the ETSEU and the WSP service provider in the PEU.

Table 5: PTCs and PCOs

WSP service provider is described in Figure 30. It connects the tester to the SUT using a TASSU connection. The tester uses *WSP TASSU Config PCO* to establish a TCP/IP connection between the TASSU Control Unit in the tester and the TASSU Router in the WAM of the Node B.

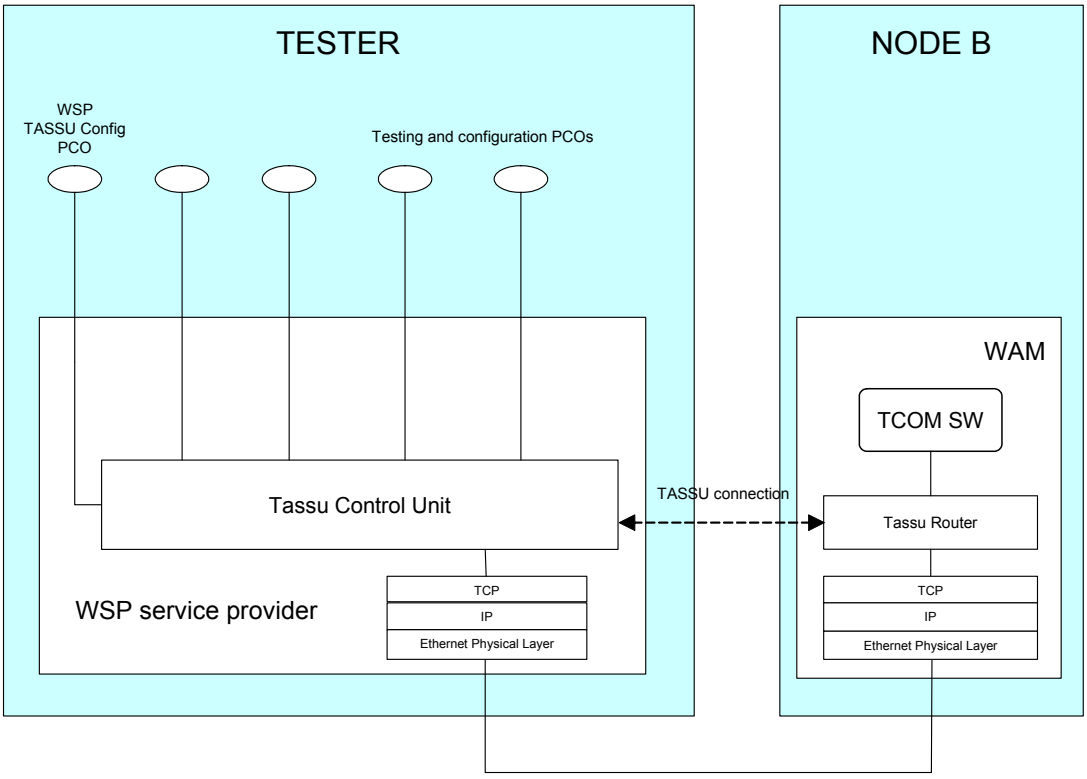

Figure 30: WSP service provider [41 p. 15]

6 TEST CASE IMPLEMENTATION AND TESTING

In this thesis system component testing test cases of WCDMA base station Telecom Software (TCOM SW) are extended to support TCOM-ATM, TCOM-WPA and TCOM-WTR interfaces. Test cases become independent of real ATM software, WPA and WTR units, since they are not needed to be present in the test configuration. Possible changes in them will not affect testing of TCOM SW. From test case logs it can be verified that the messages TCOM SW is sending to ATM software, WPA and WTR are correct and contain right parameter values. Earlier automatic checking of the message contents was not possible.

6.1 Used methods

Test case implementation is based on black box concept. The message interfaces of TCOM software are well specified, but its internal operation is not taken into account. Test cases are written in TTCN-2 with Telelogic Tau editor. They are compiled into C-language executables and executed against a real Node B by using Nokia proprietary Node B TTCN tester.

Planning of the test cases started in January 2004. Implementation started in February 2004 and was continued in parallel with testing of the test cases until May 2004. By the end of May 2004 all required test cases had been implemented and verified.

Figure 31 describes process of the TTCN test case development. TestMan database contains test plans and test case specifications. Test plans define what features of the Node B are tested, test case specifications define testing of some feature in detail. When it is decided that the Node B TTCN tester is used to test a particular feature of Node B, the tester is updated to be capable of such testing. Test case specifications are produced by Node B software testing team. From the specifications those suitable for Node B Tester are selected for TTCN implementation. After implementation the test cases are reviewed and then executed against Node B. Because in practice it is hard to produce a perfect test case in the first try, test case faults could be reported by the Node B software testing team. If they are received, corrections are made into TTCN code and a new

version of the test case is released. This iterative process continues until the test case itself does not cause errors in the test execution. Errors related to Node B software are taken care of by the Node B software testing team.

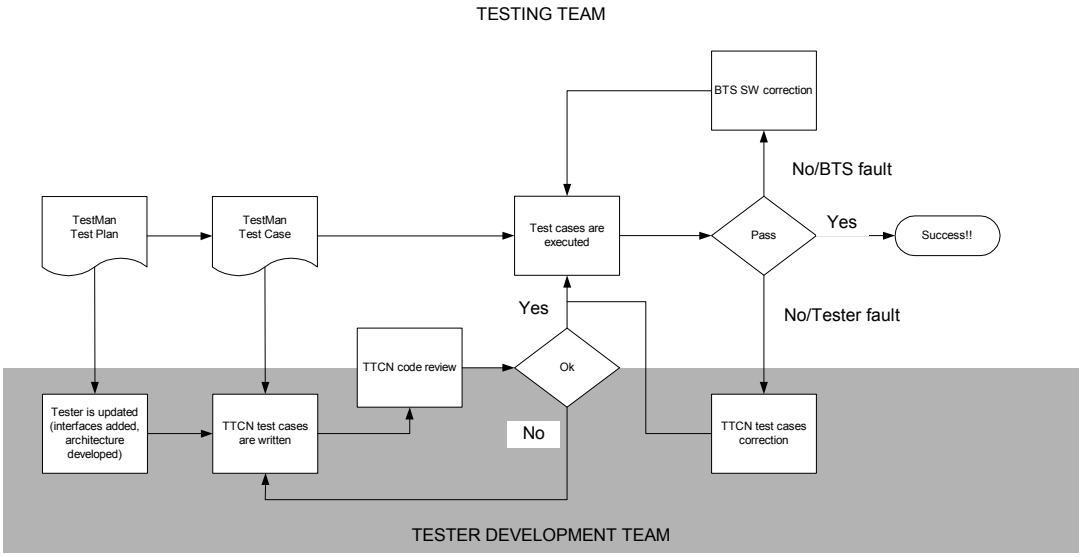

Figure 31: Test case development process [42]

6.2 Test case selection

Message sequences between TCOM SW and different subsystems are described in the test case specification. From those specifications the ones were selected which contained messages for one or more of the following interfaces: TCOM-ATM, TCOM-WPA and TCOM-WTR.

In test case specification there are given preconditions, which must be met before the test case can be executed. For selected test cases also the precondition test cases had to be modified to work with the new tester configuration. This usually meant modifying the NBAP signaling in the test case to use TASSU Iub instead of ATM Iub.

Because of the tight schedule the test cases were given priorities so that the most important test cases would be ready first. Those test cases, which were common preconditions to other cases, were needed first. TCOM-WPA and TCOM-WTR interfaces had higher priority and were introduced before TCOM-ATM.

6.3 Implementation

Test case is implemented according to test specification, which gives the following information:

- Preconditions
    - Things that must be done before the test case can be executed.
    - Usually a list of other test cases, which must be successfully executed.
- Required messages
    - Sequence of messages that are exchanged between Node B tester and TCOM SW.
    - Tester interface, which should be used for receiving or sending each message.
    - Sometimes message parameters essential to the test case.
- Post conditions
    - The conditions that the test execution must meet so that the test case can be assigned a verdict PASS.

There already existed test cases, which implemented the TCOM-DSP and NBAP signaling described in the test specifications. It was natural to use these as a starting point. Existing versions were left in the test suite and new versions with new names were created. This gives more options for TCOM SW testers in the testing phase, since the same test case is available for different test configurations. For example, if real ATM software is not available in the SUT, a version of test case can be executed, in which the tester simulates ATM software. Later when ATM software is available, the same test case could be run without ATM simulation.

The test case, which was to be upgraded was copied and given a new name. In the first phase TCOM-WPA and TCOM-WTR interfaces were added and TCOM-ATM was left out. In later phase also TCOM-ATM was added and again a new version of the test case was created.

6.3.1 WPA and WTR interface

The tester uses WPAWTR PCO to send and received messages for both WPA and WTR. In TTCN code this means that one test step must at the same time handle message sequences for both interfaces. Messages from WPA and WTR cannot arrive at the same time, but the exact order of the messages is unpredictable. In test case specification message sequences are clearly defined for both interfaces, but in practice messages could arrive in different order, and the order could change between different executions of the test case. The reason for this is that when TCOM SW initiates a procedure, which needs both WPA and WTR, it sends messages to both units sequentially. For example, an initialization message could be sent to WPA first and immediately after that to WTR. Due to the nature of the network connection between the units, the time for the messages to arrive at target units might vary a little bit, so it would possible for WTR to receive initialization message before WPA. This kind of variation is not significant in a real environment with independent WPA and WTR units, but it has to be taken into account when implementing test cases for the TTCN tester.

6.3.2 ATM interface

Adding ATM interface caused more changes in the test cases than adding WPA and WTR interfaces. When the tester is handling ATM messages the real ATM software must be disabled in the Node B. When ATM software is out of operation, the Node B cannot use ATM Iub for NBAP signaling. Therefore NBAP must be transported some other way, and in this case the TASSU interface is used. NBAP PDUs are transported inside TASSU messages.

In the existing test cases NBAP was handled through real Iub link. The cases were modified to use TASSU Iub instead. In the TTCN test suite there already existed a great number of NBAP PDU constraints for earlier test cases, the goal was that they could be used as such with the new interface. They have already been proven in earlier testing and creating new versions would increase the test suite size. Also maintenance would come more complicated, if there would be a change in one NBAP PDU constraint, the

change should be done in the other version too. By using the same constraint for both configurations (ATM Iub and TASSU Iub) this redundancy is eliminated.

Two wrappers (see section 5.1.3 for description of wrapper constraints) were created, one for sending an NBAP PDU with TASSU and one for receiving. The send constraint takes NBAP PDU as a parameter and wraps it into a TASSU PDU with correct API header. The API header contains the address of the TCOM SW task, which is responsible for receiving messages from RNC. The receive constraint is similar, the difference is in the API header values. The values are set to check that the PDU is coming from the TCOM task, which is sending messages to the RNC.

All the test steps handling Iub interface were replaced with new versions. Test step parameters for NBAP PDUs were preserved, so that they would be as similar to earlier versions as possible. The differences in the parameters are the PCOs and connection ID. For ATM Iub version the PCO was Iub_NBAP_PCO, in the new version it is NBAP_TASSU_PCO. The connection ID in the old version was the AAL ID, in the new version it is the TASSU connection ID. Send and receive statements were modified to use the correct PCO and TASSU definition. The use of NBAP PDU wrappers simplified TTCN code.

The possibility of simplifying the TTCN code in the test suite for NBAP signaling to better suite both ATM Iub and TASSU Iub was considered. In the test cases that have been developed earlier the NBAP PDU constraints are filled with parameter values in a low level of the test case ("low level" refers to the leaves of TTCN behavior tree described in Figure 19). In other words the test step, which is sending to or receiving messages from a PCO has a long list of parameters specific to some NBAP procedure. Ideally the test step would only have two parameters related to data it is supposed to handle. First the PDU that it must send to a PCO, second the PDU that it must receive in response. When the values of the PDU are set in the higher level of the test case, the lower level sending and receiving steps can be kept as simple as possible. Also it is easier to create variants of send and receive steps, like in this case would be for ATM Iub and TASSU Iub.

Another way to simplify test suite would be to hard code the TTCN constraints for NBAP PDUs. Now a constraint usually has a number of parameters, which must be filled in all the test cases in which the constraint is used. If there are two variants of the test case, maintenance of the test cases requires more work. For example, if one test case uses ATM Iub for transmitting NBAP messages and another uses TASSU Iub, and they both use the same NBAP PDU constraint, then the PDU constraint must be updated separately for both in case there are changes in the parameters. If the PDU constraint would hard coded, the change would immediately affect both test case variants. The disadvantage of hard coding the PDUs is that they cannot be reused in other test cases anymore. If two test cases for the same NBAP procedure differ in only one parameter value, they would require two different TTCN constraints. What is won in maintenance of the TTCN test steps is lost in maintenance of TTCN constraints. For example a change in the PDU definition would require going through all the hard coded TTCN constraints instead of one parameterized constraint.

This kind of reconstruction of the test steps would have taken too much time because of the large number of existing test steps. Even greater concern was that all the earlier test cases, which would have been affected by the change in PDUs, would have to be re-tested. Without testing there is no guarantee that the changes have no unwanted side effects. Re-testing was completely out of question because of the schedule.

During implementation it turned out that when ATM software is out of operation, the Node B internal message handling is affected. Normally if a Node B internal message needs to be received in the Tester instead of some Node B unit, the TASSU Router has been configured to route the message to the tester based on the receiver task field in the API header. This kind of rerouting is used to direct TCOM-DSP, TCOM-WPA and TCOM-WTR messages to the tester. The same was initially tried with TCOM-RNC messages, but it did not work because ATM SW was not initialized. Closer investigation of Node B internal logs indicated that the TCOM-RNC message is not sent like it would be when the ATM software is initialized. TASSU Router did not have anything to reroute to the tester. An alternative way to receive Node B internal

messages was used, TASSU monitoring. TASSU Test Support (TTS) was set to monitor TCOM-RNC PDU and send a copy of it to the tester. This method worked and TASSU message containing an NBAP PDU was successfully received in the tester. The monitoring method is normally suitable only for observing Node B internal messaging, because real units or software receive the messages reply to them. In this case it is safe for the tester to reply to messages received with monitoring because the ATM software is out of operation and is not reacting to those messages in any way.

Another side effect of ATM SW being out of operation is that communication between TCOM SW in different WAM units is impossible. This meant that test cases, which tested multiple WAM units, were not possible to test and they were not implemented. Technically it would be possible for the tester to act as a mediator for those messages, but it would require a new tester interface and add a lot of complexity to the test cases.

Parameter values for TCOM-ATM messages were mostly unknown in the beginning. They were not specified in the test case specification and only few values could be derived from other messages of the same test case. Known values were mainly ID values for transport channels, which had the same kind of parameter in the NBAP message. There were many parameters in the messages and value ranges of parameters were large, making it impossible to make any good assumptions about the right values for the test cases. The best way to get the values was to observe what the real ATM software uses in its communication with TCOM SW. For this purpose the test case was executed with the ATM software in operation in the Node B. Before the test case, TTS was set to monitor TCOM-ATM messages and forward a copy of them to the tester. A test step was created to decode all the TCOM-ATM messages in the WAM_ATM_PTC. After test case execution the parameter values could be collected from the tester's WAM_ATM_PTC log file. This was also a good way to verify that the new ATM PTC was able to receive and decode TCOM-ATM messages.

6.4 Implementation of an example test case

This chapter describes implementation of an example test case. In the test case a cell is created and common transport channels FACH, RACH and PCH are created in that cell. Finally FACH, RACH and PCH are activated. The message sequence chart of the test case is shown in Figure 32.

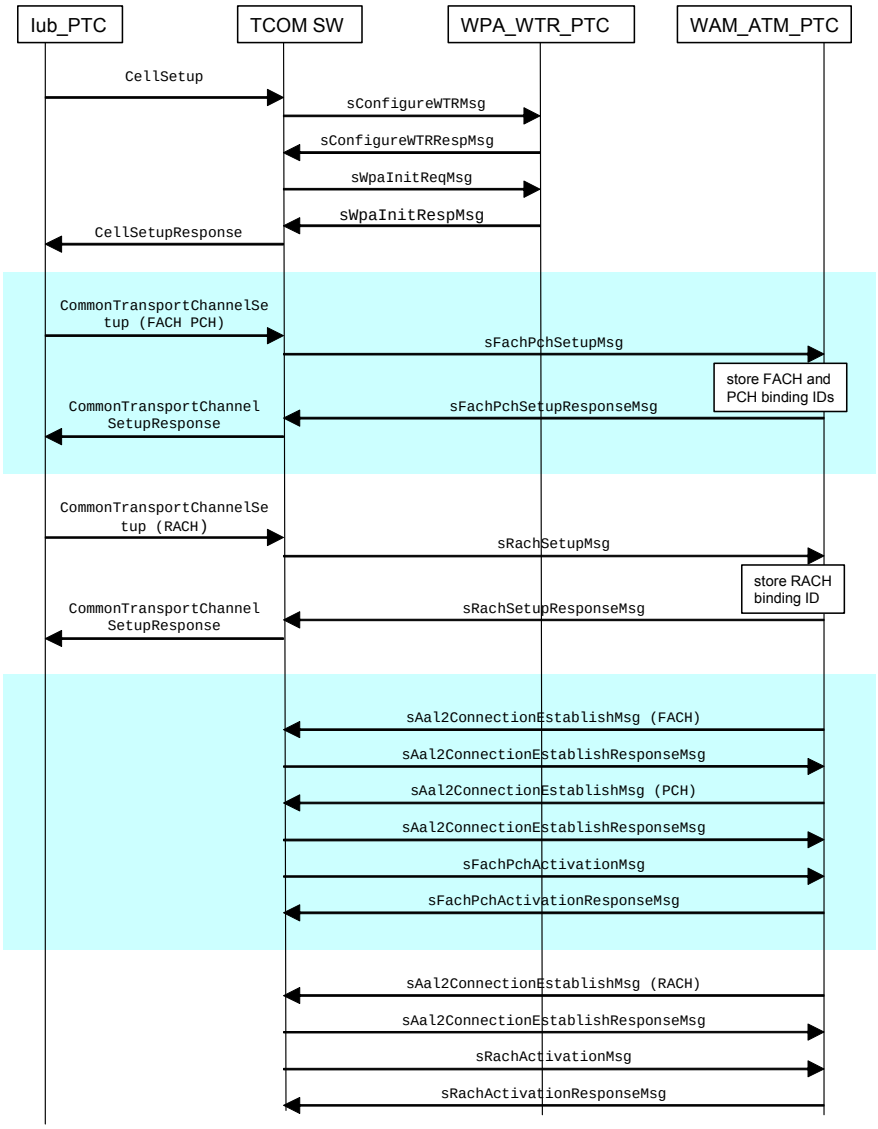

Figure 32: Test case message sequence

During cell setup the TCOM SW configures WPA and WTR units and sends initialization messages to them. The units must acknowledge the initialization messages.

For this message exchange the TCOM-WPA and TCOM-WTR interfaces are needed. When common transport channels are created the TCOM SW must inform ATM software about the new channels. TCOM SW sends setup requests to ATM software, which acknowledges them. After that ATM software requests TCOM SW to create AAL connection for the transport channels with RNC. For the channel setup and AAL establish request the TCOM-ATM interface is needed.

First thing to do is to convert signaling for NBAP procedure *CellSetupRequest* to use TASSU Iub instead of ATM Iub. New TTCN test step is created, which has the same parameters for NBAP PDU as the ATM Iub version. The difference comes in PCO and connection ID parameters. TASSU Iub version uses NBAP_TASSU_PCO instead of Iub_NBAP_PCO. The test step sends *CellSetupRequest* PDU and receives *CellSetupResponse* PDU using the wrappers described in chapter 6.3.2.

When the tester sends *CellSetupRequest* to Node B the TCOM SW starts configuring sub units. These include WPA and WTR. A test step is added for handling TCOM-WPA and TCOM-WTR signaling during cell setup. One test step is sufficient, because the tester uses the same PCO for both interfaces. The test step must receive initialization messages for WPA and WTR and must acknowledge them. A timer is started in the beginning of the test step and it is used to set the time limit how long the messages are waited. If the timer expires, test step is assigned a TTCN verdict FAIL. The same verdict also assigned if anything else than the two expected initialization messages (one for WPA, the other for WTR) with correct parameter values is received.

Normally in TTCN a test step is expecting inputs and outputs to happen in a certain order, any exception will result into a FAIL verdict. After this the test step is terminated and test engineer is not able to see what messages would have been received after the error. Test case specifications define sequence of messages expected in the test case. As was explained in section 6.3.1, handling two independent interfaces in the same PTC requires test steps to take into account that message sequences might not be exactly the same as in the test specification. Test case message sequences are made flexible by using TTCN test case variables. Variables are used to keep track of what messages have

been received, and when all the required messages are received, PASS verdict can be set.

In the CellSetup test step two variables were used, v_Test1 and v_Test2. The former is set to value 1 when initialization for WTR is received from TCOM SW, the latter gets value 1 when initialization for WPA is received. The GOTO command of TTCN is used to jump in the beginning of the test step after one of the messages has been received and the counter has been increased. PASS verdict is assigned when the counter variables have correct values (v_Test1=1 and v_Test2=1). Sequence is shown in Figure 33. When message sequence is needed to check, it can be done with the variables. In the test case specification for example test case WTR should receive message before WPA. Checking of this can be implemented by adding a condition that the variable v_Test2 must have value 0 when WTR initialization is received.

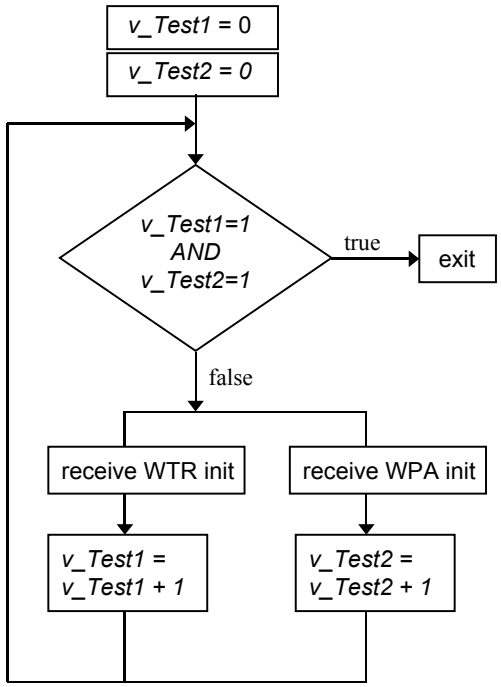

Figure 33: Concept of TTCN counter variables

If TCOM SW can successfully configure the required units for cell setup, it sends *CellSetupResponse* to the tester. Test case then proceeds to setting common transport channels FACH, PCH and RACH into the cell that was just created.

Implementation of TASSU Iub test step for *CommonTransportChannelSetup* is similar to implementation of *CellSetupRequest*. The difference is in the NBAP procedure, *CommonTransportChannelSetup* is sent and *CommonTransportChannelSetupResponse* is received instead of *CellSetupRequest* and *CellSetupResponse*. Also the parameters for NBAP PDUs are different and specific to the NBAP procedures. The general structure of test steps is the same.

When TCOM SW receives *CommonTransportChannelSetup* from the tester, channel setup requests are sent to ATM software. In the example test case TCOM sends first a combined setup request for FACH and PCH, and after that a setup request for RACH. ATM software must acknowledge both. A test step is added for handling both setups. Counter variables described for WPA and WTR test step are used also here. Setup messages contain important information for the later part of the test case, the channel activation phase. This information is the binding ID values for FACH, PCH and RACH, which are extracted from the setup messages and stored into TTCN test suite variables. If channel setup requests are successful, TCOM SW sends *CommonTranportChannelSetupResponse* to the tester.

FACH, PCH and RACH are then activated. Tester sends two AAL connection establish request messages to TCOM SW, the first contains binding ID of FACH and the other binding ID of PCH. This is the scenario when ATM software has created ATM connections with RNC for new transport channels (FACH and PCH) and TCOM SW can make the channels active. TCOM SW acknowledges the AAL connection establish request messages and activates channels with FACH PCH activation message, which the tester acknowledges. Procedure is the same for RACH activation. Tester sends AAL connection establish with the binding ID of RACH. TCOM SW acknowledges it and sends RACH activation message, which the tester acknowledges.

6.5 Testing of the example test case

Execution of an example test case is presented in this section. Example covers test steps from starting the tester up to setting up common transport channels in the Node B.

Excerpts of test case logs from MASTER_Tester, WPAWTR_PTC_1 and WAM_ATM_PTC_1 are presented. The logs have been captured during test case execution against real WCDMA base station.

Before the test case execution the test environment has been started. ETSEU and PEU have been started and the Node B has been powered on. The first test is *InitPlatformP4*, which sets up tester's internal connections and creates a TCP/IP connection between the tester and the WAM board in the Node B.

```
: test
: MASTERTester ts_invokereq (default = sendNow(D))
[[MASTER_Tester]]* * InitPlatformP4 begun at 11:18:08:592, May 18, 2004
. . .
[[MASTER_Tester]]V Verdict PASS
[[MASTER_Tester]]. End InitPlatformP4    @+1,461
[[MASTER_Tester]]* * InitPlatformP4 ended at 11:18:10:53, May 18, 2004
```

Next, the tester must bring the Node B to a state, where it can accept NBAP messages. This is done with the test case *BtsSwStartupNoFaultInd1P4*. This means that the tester must exchange the same messages with the TCOM SW that the TCOM SW would handle during start up of the complete Node B. Complete here means that all units and SW would be present. In system component testing of WAM the tester handles the messages from missing units or SW. The exact sequence of Node B start up test case is out of scope of this thesis. The text "NoFaultInd" in the name means that this start up test case will not display Fault messages in the test case logs. With Fault messages the Node B informs about many kinds of error situations. Because in the system component testing a lot of HW or SW components are not present, the Node B might send a lot of Fault messages. They are filtered out to prevent them from disturbing the test case execution and flooding the test case logs.

One important thing to do in the start up test case is to configure TASSU connection so that all the required messages are available to the Tester. TASSU Router in the WAM board is configured to route messages directed to WPA and WTR units to the tester. TASSU Test Support is configured to monitor messages that contain NBAP PDUs. Due

to Node B internal architecture, these messages cannot be routed to the tester, but they can be monitored instead.

```
:test
: MASTERTester ts_invokereq (default = sendNow(D))
[[MASTER_Tester]]* * BtsSwStartupNoFaultInd1P4 begun at 11:18:17:19, May 18, 2004
. . .
[[MASTER_Tester]]V Verdict PASS
[[MASTER_Tester]]. End BtsSwStartupNoFaultInd1P4    @+9,375
[[MASTER_Tester]]* * BtsSwStartupNoFaultInd1P4 ended at 11:18:26:393, May 18, 2004
```

After the start up test case the Node B is in a state, where it is ready to accept NBAP messages. The first thing the Node B does is that it starts sending NBAP message *Reset* to RNC. The purpose of *Reset* is that the RNC gets informed about Node B start up. The RNC can now configure Node B.

Test case *Reset_Initiated_by_the_Node_B_and_Startup_Audit_P4* receives *Reset* message from the Node B and replies to it with NBAP message *ResetResponse*. The Node B stops sending *Reset*. Next in the test case NBAP message *AuditRequest* is sent to the Node B. With this message the RNC queries the Node B capabilities and current status. The Node B replies with *AuditResponse*.

```
: test
: MASTERTester ts_invokereq (default = sendNow(D))
[[MASTER_Tester]]* * Reset_Initiated_by_the_Node_B_and_Startup_Audit_P4 begun at
11:18:28:422, May 18, 2004
. . .
[[MASTER_Tester]]V Verdict PASS
[[MASTER_Tester]]. End Reset_Initiated_by_the_Node_B_and_Startup_Audit_P4    @+7,172
[[MASTER_Tester]]* * Reset_Initiated_by_the_Node_B_and_Startup_Audit_P4 ended at
11:18:35:593, May 18, 2004
```

When NBAP procedures *Reset* and *Audit* have been done, cells can be set up in the Node B. For each cell common transport channels RACH, FACH and PCH must be set up. This is done in the test case *CellSetup_CommonTransportChannelSetup_P4*. During cell setup the WPA and WTR units are initialized. During common transport channel setup ATM resources are reserved for new channels. These are described in more detail in the WPAWTR_PTC_1 and WAM_ATM_PTC_1 logs below.

```
:test
```

```
: MASTERTester ts_invokereq (default = sendNow(D))
[[MASTER_Tester]]* * CellSetup_CommonTransportChannelSetup_P4 begun at 11:18:45:575, May
18, 2004
. . .
[[MASTER_Tester]]V Verdict PASS
[[MASTER_Tester]]. End CellSetup_CommonTransportChannelSetup_P4    @+17,169
[[MASTER_Tester]]* * CellSetup_CommonTransportChannelSetup_P4 ended at 11:19:15:273, May
18, 2004
```

When the test case *CellSetup_CommonTransportChannelSetup_P4* is started in the MASTER tester, a test step *s_wpa_wtr_CellSetup_MasterWAM_P1_MT* is started in the parallel test component for WPA and WTR. It handles messages that the TCOM SW would send to real units. WTR unit is initialized with message *sConfigureWTRMsg*, it must be acknowledged with *sConfigureWTRRespMsg*. WPA unit is initialized with message *sWpaInitReqMsg* and it is acknowledged with *sWpaInitRespMsg*.

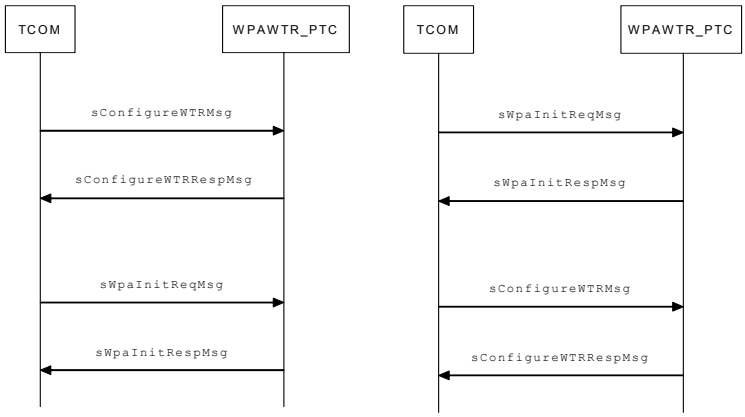

Figure 34: Two possible message sequences of WPA and WTR initialisation

Because messages from two independent interfaces are handled in the same PTC, the test case must be flexible and not expect WPA and WTR messages to arrive in certain order. Test case variable approach described in the chapter 6.4 has been used. Timer *T_WPA_WTR* is started in the beginning of the test step. The test step exits and assigns a FAIL verdict if both messages have not been received within 30 seconds.

```
[[WPAWTR_PTC_1]]* * s_wpa_wtr_CellSetup_MasterWAM_P1_MT begun at 11:18:46:120, May 18,
2004
[[WPAWTR_PTC_1]]B Begin s_wpa_wtr_CellSetup_MasterWAM_P1_MT    @49,970
```

```
[[WPAWTR_PTC_1]]A   v_Test1 := 0
[[WPAWTR_PTC_1]]A   v_Test2 := 0
[[WPAWTR_PTC_1]]Z   START T_WPA_WTR(30,0)    @+0,3
. . .
[[WPAWTR_PTC_1]]A   v_Test1 := 1
[[WPAWTR_PTC_1]]r   WPAWTR_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+1,636
[[WPAWTR_PTC_1]]r     tassu_dataind {
[[WPAWTR_PTC_1]]r       sapId 1,
[[WPAWTR_PTC_1]]r       tassu_dut_pdu apiMsg wpaWtrPdu sConfigureWTRMsg {
. . .
[[WPAWTR_PTC_1]]s   WPAWTR_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+1,685
[[WPAWTR_PTC_1]]s     tassu_datareq {
[[WPAWTR_PTC_1]]s       sapId 1,
[[WPAWTR_PTC_1]]s       tassu_dut_pdu apiMsg wpaWtrPdu sConfigureWTRRespMsg {
. . .
[[WPAWTR_PTC_1]]A   v_Test2 := 1
[[WPAWTR_PTC_1]]r   WPAWTR_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+1,693
[[WPAWTR_PTC_1]]r     tassu_dataind {
[[WPAWTR_PTC_1]]r       sapId 1,
[[WPAWTR_PTC_1]]r       tassu_dut_pdu apiMsg wpaWtrPdu sWpaInitReqMsg {
. . .
[[WPAWTR_PTC_1]]s   WPAWTR_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+1,728
[[WPAWTR_PTC_1]]s     tassu_datareq {
[[WPAWTR_PTC_1]]s       sapId 1,
[[WPAWTR_PTC_1]]s       tassu_dut_pdu apiMsg wpaWtrPdu sWpaInitRespMsg {
. . .
[[WPAWTR_PTC_1]]A   R := pass
[[WPAWTR_PTC_1]]X Exit s_wpa_wtr_CellSetup
[[WPAWTR_PTC_1]]V Verdict PASS
```

CellSetup_CommonTransportChannelSetup_P4 starts test step *s_atm_SCCPCH_Setup_MT* in the parallel test component for ATM. The test step handles messages that TCOM SW sends to ATM software when common transport channel FACH is set up. When a new channel is set up, ATM transport resources must be reserved for the channel. FACH and PCH channels are set up in the same message *sFachPchSetupMsg*. The message contains binding ID values for both channels, which must be stored for later use. FACH binding ID is stored into test case variable *v_atm_fach_bindingId*, PCH binding ID is stored into *v_atm_pch_bindingId*. Request message is acknowledged with *sFachPchSetupResponseMsg*. Timer *T_ATM* is used to enable test step termination if the message is not received in the PTC.

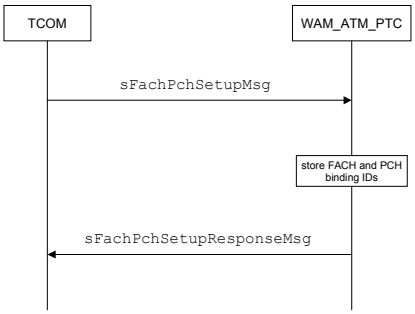

Figure 35: Message sequence of Fach and Pch setup

```
[[WAM_ATM_PTC_1]]* * s_atm_SCCPCH_Setup_MT begun at 11:19:05:0, May 18, 2004
[[WAM_ATM_PTC_1]]B Begin s_atm_SCCPCH_Setup_MT    @70,7
[[WAM_ATM_PTC_1]]A   v_TASSU_SapId := 1

[[WAM_ATM_PTC_1]]Z   START T_ATM(30,0)    @+0,2
. . .
[[WAM_ATM_PTC_1]]A   v_atm_fach_bindingId := 1
[[WAM_ATM_PTC_1]]A   v_atm_pch_bindingId := 2
[[WAM_ATM_PTC_1]]r   WAM_ATM_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+2,219
[[WAM_ATM_PTC_1]]r     tassu_dataind {
[[WAM_ATM_PTC_1]]r       sapId 1,
[[WAM_ATM_PTC_1]]r       tassu_dut_pdu apiMsg tcomAtmPdu sFachPchSetupMsg {
. . .
[[WAM_ATM_PTC_1]]s   WAM_ATM_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+3,78
[[WAM_ATM_PTC_1]]s     tassu_datareq {
[[WAM_ATM_PTC_1]]s       sapId 1,
[[WAM_ATM_PTC_1]]s       tassu_dut_pdu apiMsg tcomAtmPdu sFachPchSetupResponseMsg {
. . .
[[WAM_ATM_PTC_1]]X Exit s_atm_FachPchSetup
. . .
[[WAM_ATM_PTC_1]]V Verdict PASS
[[WAM_ATM_PTC_1]]C   CANCEL T_ATM    @+3,87
```

Test step *s_atm_PRACH_Setup_MT* is similar to previous *s_atm_SCCPCH_Setup_MT,* here resources for common transport channel RACH are reserved. TCOM sends message *sRachSetupMsg* containing the binding ID. It is stored into test case variable *v_atm_rach_bindingId*, and the request message is acknowledged with *sRachSetupResponseMsg*.

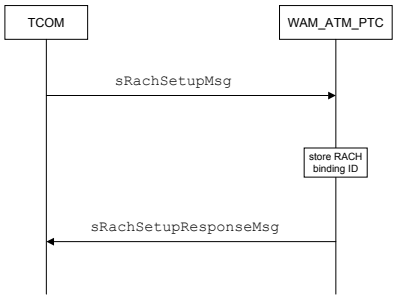

Figure 36: Message sequence of Rach setup

```
[[WAM_ATM_PTC_1]]* * s_atm_PRACH_Setup_MT begun at 11:19:08:640, May 18, 2004
[[WAM_ATM_PTC_1]]B Begin s_atm_PRACH_Setup_MT    @73,647
[[WAM_ATM_PTC_1]]A   v_TASSU_SapId := 1
[[WAM_ATM_PTC_1]]N Enter s_atm_RachSetup
[[WAM_ATM_PTC_1]]Z   START T_ATM(30,0)    @+0,4
. . .
[[WAM_ATM_PTC_1]]A   v_atm_rach_bindingId := 3
[[WAM_ATM_PTC_1]]r   WAM_ATM_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+2,84
[[WAM_ATM_PTC_1]]r     tassu_dataind {
[[WAM_ATM_PTC_1]]r       sapId 1,
[[WAM_ATM_PTC_1]]r       tassu_dut_pdu apiMsg tcomAtmPdu sRachSetupMsg {
. . .
[[WAM_ATM_PTC_1]]s   WAM_ATM_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+2,206
[[WAM_ATM_PTC_1]]s     tassu_datareq {
[[WAM_ATM_PTC_1]]s       sapId 1,
[[WAM_ATM_PTC_1]]s       tassu_dut_pdu apiMsg tcomAtmPdu sRachSetupResponseMsg {
. . .
[[WAM_ATM_PTC_1]]A   R := pass
[[WAM_ATM_PTC_1]]X Exit s_atm_RachSetup
. . .
[[WAM_ATM_PTC_1]]V Verdict PASS
[[WAM_ATM_PTC_1]]C   CANCEL T_ATM    @+2,212
```

After a common transport channel has been set up, it is activated. This is done when ATM connection specific to that channel has been established between the Node B and the RNC. The tester simulates creation of the connection by sending *sAal2ConnectionEstablishMsg* to the TCOM SW. The message must contain the binding ID that was received in the set up message. Binding Ids for RACH, FACH and PCH were stored into test case variables and they are used in the connection establish messages. TCOM SW replies with *sAal2ConnectionEstablishResponseMsg* and sends channel activation message, which is acknowledged with activation response message.

Test step *s_atm_FACH_Activation_MT* activates channels FACH and PCH. First *sAal2ConnectionEstablishMsg* is sent containing binding ID from the variable *v_atm_fach_bindingId*. Response *sAal2ConnectionEstablishResponseMsg* is received and *sAal2ConnectionEstablishMsg* is sent again, this time containing binding ID from the variable *v_atm_pch_bindingId*. Response *sAal2ConnectionEstablishResponseMsg* is received. Now TCOM is informed that connection for both channels has been established. TCOM sends *sFachPchActivationMsg*, which is answered with *sFachPchActivationResponseMsg*.

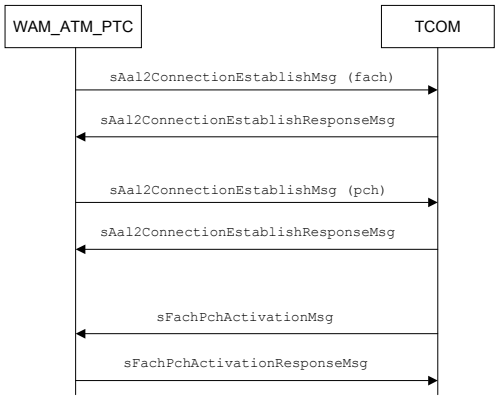

Figure 37: Message sequence of Fach and Pch activation

```
[[WAM_ATM_PTC_1]]* * s_atm_FACH_Activation_MT begun at 11:19:12:270, May 18, 2004
[[WAM_ATM_PTC_1]]B Begin s_atm_FACH_Activation_MT    @77,277
[[WAM_ATM_PTC_1]]A   v_TASSU_SapId := 1
[[WAM_ATM_PTC_1]]N Enter s_atm_FachPchActivation
[[WAM_ATM_PTC_1]]N Enter s_atm_Aal2ConnectionEstablishMsg
[[WAM_ATM_PTC_1]]s   WAM_ATM_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+0,4
[[WAM_ATM_PTC_1]]s     tassu_datareq {
[[WAM_ATM_PTC_1]]s       sapId 1,
[[WAM_ATM_PTC_1]]s       tassu_dut_pdu apiMsg tcomAtmPdu sAal2ConnectionEstablishMsg {
. . .
[[WAM_ATM_PTC_1]]s         bindingId 1,
. . .
[[WAM_ATM_PTC_1]]r   WAM_ATM_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+0,37
[[WAM_ATM_PTC_1]]r     tassu_dataind {
[[WAM_ATM_PTC_1]]r       sapId 1,
[[WAM_ATM_PTC_1]]r       tassu_dut_pdu apiMsg tcomAtmPdu
sAal2ConnectionEstablishResponseMsg {
. . .
[[WAM_ATM_PTC_1]]s   WAM_ATM_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+0,44
[[WAM_ATM_PTC_1]]s     tassu_datareq {
```

```
[[WAM_ATM_PTC_1]]s       sapId 1,
[[WAM_ATM_PTC_1]]s       tassu_dut_pdu apiMsg tcomAtmPdu sAal2ConnectionEstablishMsg {
. . .
[[WAM_ATM_PTC_1]]s         bindingId 2,
. . .
[[WAM_ATM_PTC_1]]r   WAM_ATM_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+0,81
[[WAM_ATM_PTC_1]]r     tassu_dataind {
[[WAM_ATM_PTC_1]]r       sapId 1,
[[WAM_ATM_PTC_1]]r       tassu_dut_pdu apiMsg tcomAtmPdu
sAal2ConnectionEstablishResponseMsg {
. . .
[[WAM_ATM_PTC_1]]r   WAM_ATM_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+0,182
[[WAM_ATM_PTC_1]]r     tassu_dataind {
[[WAM_ATM_PTC_1]]r       sapId 1,
[[WAM_ATM_PTC_1]]r       tassu_dut_pdu apiMsg tcomAtmPdu sFachPchActivationMsg {
. . .
[[WAM_ATM_PTC_1]]s   WAM_ATM_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+0,189
[[WAM_ATM_PTC_1]]s     tassu_datareq {
[[WAM_ATM_PTC_1]]s       sapId 1,
[[WAM_ATM_PTC_1]]s       tassu_dut_pdu apiMsg tcomAtmPdu sFachPchActivationResponseMsg {
. . .
[[WAM_ATM_PTC_1]]X Exit s_atm_FachPchActivation
[[WAM_ATM_PTC_1]]V Verdict PASS
```

Test step *s_atm_RACH_Activation_MT* does the same sequence for RACH channel. *sAal2ConnectionEstablishMsg* is sent with the binding ID from test case variable *v_atm_rach_bindingId*. *sAal2ConnectionEstablishResponseMsg* is received as a reply, and immediately after that *sRachActivationMsg*. Finally *sRachActivationResponseMsg* is sent to conclude the common transport channel setup and activation.

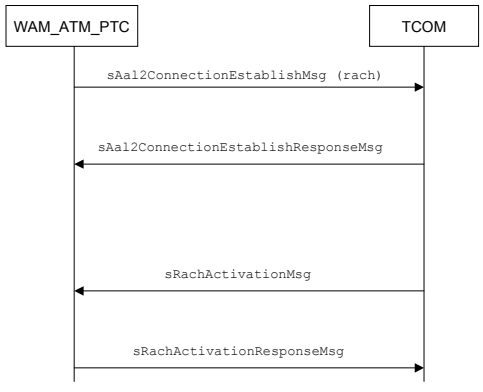

Figure 38: Message sequence of Rach activation

```
[[WAM_ATM_PTC_1]]* * s_atm_RACH_Activation_MT begun at 11:19:13:951, May 18, 2004
```

```
[[WAM_ATM_PTC_1]]B Begin s_atm_RACH_Activation_MT    @78,958
. . .
[[WAM_ATM_PTC_1]]s   WAM_ATM_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+0,4
[[WAM_ATM_PTC_1]]s     tassu_datareq {
[[WAM_ATM_PTC_1]]s       sapId 1,
[[WAM_ATM_PTC_1]]s       tassu_dut_pdu apiMsg tcomAtmPdu sAal2ConnectionEstablishMsg {
. . .
[[WAM_ATM_PTC_1]]s         bindingId 3,
. . .
[[WAM_ATM_PTC_1]]r   WAM_ATM_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+0,40
[[WAM_ATM_PTC_1]]r     tassu_dataind {
[[WAM_ATM_PTC_1]]r       sapId 1,
[[WAM_ATM_PTC_1]]r       tassu_dut_pdu apiMsg tcomAtmPdu
sAal2ConnectionEstablishResponseMsg {
. . .
[[WAM_ATM_PTC_1]]r   WAM_ATM_TASSU_PCO_1?WSP_TASSU_DATA_PRIM a_TASSU_DataInd    @+0,131
[[WAM_ATM_PTC_1]]r     tassu_dataind {
[[WAM_ATM_PTC_1]]r       sapId 1,
[[WAM_ATM_PTC_1]]r       tassu_dut_pdu apiMsg tcomAtmPdu sRachActivationMsg {
. . .
[[WAM_ATM_PTC_1]]s   WAM_ATM_TASSU_PCO_1!WSP_TASSU_DATA_PRIM a_TASSU_DataReq    @+0,139
[[WAM_ATM_PTC_1]]s     tassu_datareq {
[[WAM_ATM_PTC_1]]s       sapId 1,
[[WAM_ATM_PTC_1]]s       tassu_dut_pdu apiMsg tcomAtmPdu sRachActivationResponseMsg {
. . .
[[WAM_ATM_PTC_1]]X Exit s_atm_RachActivation
[[WAM_ATM_PTC_1]]V Verdict PASS
```

6.6 Problems in test case implementation

There were difficulties to get NBAP signaling work over TASSU connection instead of ATM Iub. A lot of experimenting with TASSU Router and TASSU Test Support had to be done before NBAP PDUs could successfully be changed between the Tester and the TCOM SW.

In general the parameter values used with the messages for ATM, WPA and WTR interfaces were unknown in the beginning. The solution was to observe what values real ATM SW, WPA and WTR use when they are active in the Node B. A test case was constructed, which triggered message exchange in TCOM-ATM, TCOM-WPA and TCOM-WTR interfaces. Those messages were monitored with the TASSU Test Support and the parameter values were reused in the test cases.

In ATM interface there was differences in some message definitions between the Tester and the TCOM SW. A test case containing such messages gave a FAIL verdict or stopped without giving any result. By interpreting test case logs it was seen that message sent by TCOM SW was not properly decoded in the tester. After the message causing error was identified, its definitions for TCOM SW and the tester were compared. In all cases the error was found in the Tester's message definition, usually it was a data type mismatch or some data field was missing. A correction request was sent to the tester platform development team, which sent a corrected message definition. The test case was executed again and the message correctness was verified.

The ATM interface contained very long messages, much longer than any message used so far with the tester. It turned out that the tester was not able to reply fast enough to those messages. When sending some requests to ATM SW, the TCOM SW starts a short timer and the ATM SW must reply before the timer expires. Otherwise the TCOM SW cancels the procedure. When the tester received a long message, the message decoding took too long and the reply was sent after TCOM had already cancelled the procedure. The cause of this problem took some time to identify, because the test case logs looked flawless, but the result in the TCOM SW was far from it. The problem was

debugged by using logs external to the Node B TTCN Tester tool. The solution was to change message decoding in the tester so that only the message fields relevant for that test case were processed, not the whole message. This speeded up message decoding significantly and the Tester was able to reply to all messages within the required time limit.

6.7 Results

In this thesis the first TTCN test cases to use new ATM, WPA and WTR interfaces of the Node B TTCN Tester were developed. NBAP signaling between the Node B and the Tester was converted to use TASSU connection instead of ATM Iub. The developed test cases have been tested against the WCDMA base station with successful results. All the new interfaces worked correctly, the Tester was able to simulate ATM SW, WPA unit and WTR unit. NBAP signaling over TASSU connection works without problems.

The TTCN test cases can be used as an example in the future development. Most important examples will be the use of TASSU Iub and handling of AAL2 connection requests. The TTCN test steps, constraints, variables and constants can directly be reused in development of new test cases.

7 CONCLUSIONS

The goal of this thesis was to produce the first TTCN test cases, which use new ATM, WPA and WTR interfaces in the Node B TTCN tester. The test cases are used in the system component testing of the TCOM SW.

The test cases developed in this thesis improve automation in the testing of TCOM SW. Testing is no more dependent on ATM software or WPA and WTR units, because the tester cases can simulate them. Verifying the correct functionality of the TCOM SW in TCOM-ATM, TCOM-WPA and TCOM-WTR interfaces is automated and detailed logs of messages in the interfaces are produced.

The new ATM, WPA and WTR interfaces in the Node B TTCN tester improve the possibilities to carry out rigorous system component testing campaigns of TCOM SW. The test cases developed in this thesis were just a beginning. Test case development continues in synchronisation with TCOM SW testing. A number of new test cases will be developed and most of them will take advantage of the new interfaces.

REFERENCES

[1] Whittaker James, “What Is Software Testing? And Why Is It So Hard?”, IEEE Software, January/February 2000. Available from http://www.computer.org/software/so2000/pdf/s1070.pdf, accessed 5.1.2005.

[2] Dustin Elfriede, Rashka Jeff, Paul John, “Automated software testing: introduction, management, and performance”, Addison Wesley Longman, 1999, 575p.

[3] Tapia José Manuel, “WCDMA BTS SW Releases: Node B Tester Maintenance Project Plan”, Nokia, 2004, version 2.0.2, 34 p. (Confidential)

[4] Pressman Roger S., “Software Engineering: A practitioner’s approach”, McGraw-Hill, 2001, 860 p.

[5] Kit Edward, “Software Testing in the Real World: Improving the Process”, Addison-Wesley, 1995, 252 p.

[6] Bach James, “Test Automation Snake Oil”, 1999, 6p. Available from http://www.satisfice.com/articles/test_automation_snake_oil.pdf, accessed 5.1.2005.

[7] Marick Brian, “When Should a Test Be Automated?”, 1998, 20 p. Available from http://www.testing.com/writings/automate.pdf, accessed 5.1.2005.

[8] Zallar Kerry, “Are You Ready for the Test Automation Game?”, article in STQE, November/December 2001.

[9] Löbbert Achim, Deichmann Volker “XENA NodeB protocol testing: Project Plan”, Nokia, 2002, version 1.8.0, 17 p. (Confidential)

[10] Kaaranen Heikki, Ahtiainen Ari, Laitinen Lauri, Naghan Siamäk, Niemi Valtteri, “UMTS Networks: Architecture, Mobility and Services”, John Wiley & Sons, 2001, 302 p.

[11] 3GPP Technical Specification 25.401, “UTRAN Overall Description”, 2004, version 6.4.0, 44 p.

[12] 3GPP Technical Specification 25.413, ”UTRAN Iu Interface RANAP Signalling (Release 6)”, 2004, version 6.3.0, 273 p.

[13] 3GPP Technical Specification 25.423, ”UTRAN Iur Interface RNSAP Signalling (Release 6)”, 2004, version 6.3.0, 618 p.

[14] 3GPP Technical Specification 25.433, “UTRAN Iub Interface NBAP Signalling (Release 6)”, 2004, version 6.3.0, 690 p.

[15] 3GPP Technical Specification 25.211, “Physical Channels and Mapping of Transport Channels onto Physical Channels (FDD)”, 2004, version 6.2.0, 51 p.

[16] Holma Harri, Toskala Antti, “WCDMA for UMTS”, John Wiley & Sons, 2000, 322 p.

[17] Grundström Mika, Mickos Roy, ”ATM-tekniikka ja Monipalveluverkot”, Suomen Atk-kustannus Oy, 1997, 396 p.

[18] Stallings William, ”Data and Computer Communications, Fifth Edition”, Prentice-Hall, 1997, 798 p.

[19] CCITT Recommendation I.321, “B-ISDN Protocol Model and Its Application”, 1991, 7 p.

[20] ITU-T Recommendation I.361, “B-ISDN ATM Layer Specification”, 1999, 34 p.

[21] ITU-T Recommendation I.362, “B-ISDN ATM Adaptation Layer (AAL) Functional Description”, 1993, 3 p.

[22] ITU-T Recommendation I.363, “B-ISDN ATM Adaptation Layer (AAL) Specification”, 1993, 96 p.

[23] ITU-T Recommendation Q.2100, “B-ISDN Signalling ATM Adaptation Layer (SAAL) Overview Description”, 1994, 3 p.

[24] 3GPP Technical Specification 25.432, “UTRAN Iub Interface: Signalling Transport”, 2001, version 4.0.0, 7 p.

[25] ITU-T Recommendation Q.2110, “B-ISDN ATM Adaptation Layer – Service Specific Connection Oriented Protocol (SSCOP)”, 1994, 95 p.

[26] ITU-T Recommendation Q.2130, “B-ISDN Signalling ATM Adaptation Layer – Service Specific Coordination Function for Support of Signalling at the User Network Interface (SSFC at UNI)”, 1994, 54 p.

[27] 3GPP Technical Specification 25.434, “UTRAN Iub Interface Data Transport and Transport Signalling for Common Transport Channel Data Streams”, 2003, version 6.0.0, 11 p.

[28] ITU-T Recommendation I.366.1, “Segmentation and Reassembly Service Specific Convergence Sublayer for the AAL Type 2”, 1998, 31 p.

[29] 3GPP Technical Specification 25.430, ”UTRAN Iub Interface: General Aspects and Principles”, 2004, version 6.2.0, 25 p.

[30] Evisalmi Ari, Herranen Vesa, Koivisto Timo, Nyyssönen Aki, “WCDMA SW: SW Architecture Specification”, Nokia, 2004, 80 p. (Confidential)

[31] Vähäkangas Juha, “Ultrasite WCDMA BTS Product Definition”, Nokia, 2002, 129 p. (Confidential)

[32] Behm Leif, Stålö Markus, Thorselius Kaj, “Ultrasite WCDMA BTS: WPA SW System Requirements Specification”, Nokia, 2001, 57 p. (Confidential)

[33] Väkevä Kari, “ATM SW: Requirement and Implementation Specification”, Nokia, 1999, 34 p. (Confidential)

[34] Open Base Station Architecture Initiative, “The Development and Benefits of an Open Base Station Architecture”, White Paper MWG-040318-006-01. Available from http://www.obsai.org/latests/publicdocuments.htm

[35] ISO/IEC International Standard 9646-3, “Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN)”, 1998, 265 p.

[36] IEC tutorial, “Tree and Tabular Combined Notation (TTCN)”. Available from http://www.iec.org/online/tutorials/ttcn/.

[37] Wiles Anthony, Randall Steve, Cousin Philippe, Thalhammer Johann, "The Relevance of Conformance Testing for Interoperability Testing", http://www. etsi.org/ptcc/ptccdownloads.htm

[38] ETSI ES 201 873-1 (V2.2.1), "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 2: TTCN-3 Tabular Presentation Format (TFT)", 2003, 178 p.

[39] Deichmann Volker, Joshi Dilip, Löbbert Achim, "XENA NodeB protocol testing: User Manual", Nokia, 2004, version 2.0.4, 84 p. (Confidential)

[40] "WCDMA BTS: TASSU – BS SW Interface Specification", Nokia, 2003, 199 p. (Confidential)

[41] Löbbert Achim, Rooney Brian, "WSPB Service Provider", Nokia, 2002, 77 p. (Confidential)

[42] Tapia, José Manuel, email discussion 17.12.2004.