Methodology for Testing RFID Applications

PhD Thesis

Andreas Hübner

De Montfort University (DMU)
Cyber Technology Institute
Leicester - United Kingdom

This thesis is submitted in partial fulfilment of the requirements for the Doctor of Philosophy (PhD), awarded by De Montfort University.

March 2018

Supervisors:
Dr. Helge Janicke
Dr. Christian Facchi

Abstract

Radio Frequency Identification (RFID) is a promising technology for process automation and beyond that capable of identifying objects without the need for a line-of-sight. However, the trend towards automatic identification of objects also increases the demand for high quality RFID applications. Therefore, research on testing RFID systems and methodical approaches for testing are needed.

This thesis presents a novel methodology for the system level test of RFID applications. The approach called ITERA, allows for the automatic generation of tests, defines a semantic model of the RFID system and provides a test environment for RFID applications. The method introduced can be used to gradually transform use cases into a semi-formal test specification. Test cases are then systematically generated, in order to execute them in the test environment.

It applies the principle of model based testing from a black-box perspective in combination with a virtual environment for automatic test execution. The presence of RFID tags in an area, monitored by an RFID reader, can be modelled by time-based sets using set-theory and discrete events. Furthermore, the proposed description and semantics can be used to specify RFID systems and their applications, which might also be used for other purposes than testing. The approach uses the Unified Modelling Language to model the characteristics of the system under test. Based on the ITERA meta model test execution paths are extracted directly from activity diagrams and RFID specific test cases are generated.

The approach introduced in this thesis allows to reduce the efforts for RFID application testing by systematically generating test cases and the automatic test execution. In combination with meta model and by considering additional parameters, like unreliability factors, it not only satisfies functional testing aspects, but also increases the confidence in the robustness of the tested application. Mixed with the instantly available virtual readers, it has the potential to speed up the development process and decrease the costs - even during the early development phases. ITERA can be used for highly automated testing, reproducible tests and because of the instantly available readers, even before the real environment is deployed. Furthermore, the total control of the RFID environment enables to test applications which might be difficult to test manually.

This thesis will explain the motivation and objectives of this new RFID application test methodology. Based on a RFID system analysis it proposes a practical solution on the identified issues. Further, it gives a literature review on testing fundamentals, model based test case generation, the typical components of a RFID system and RFID standards used in industry.

Declaration

I hereby declare that the work described in this thesis is original work undertaken by me between October 2011 and April 2018 for the degree of Doctor of Philosophy, at the Zentrum für Angewandte Forschung, Technische Hochschule Ingolstadt, Germany and the Cyber Technology Institute, Department of Computer Science and Engineering (CSE), De Montfort University, United Kingdom.

In conformance to the Student Declaration Form of the De Montfort University:

- I declare that the work was solely conducted during registration for the above award with De Montfort University, under University supervision.
- I declare that no material contained in the thesis has been used in any other submission for an academic award.
- I confirm that the work represented in this submission was undertaken solely by myself except where otherwise attributed.
- I acknowledge the copyright and other intellectual property rights in relation to my thesis and other work prepared and submitted in the course of my studies shall belong to the University, except where specifically agreed otherwise by the University in writing.
- I certify that I have followed all requirements for ethical approval as specified by my Faculty Research Ethics Committee.

Ingolstadt, 31.03.2018 *Andreas R. Hübner*

Publications

The following list of Publications originated during the work on this topic:

1. **Rifidi Toolkit: Virtuality for Testing RFID**
   presented at the International Conference on Systems and Networks Communications 2012
2. **Automated Test Code Generation Based on Formalized Natural Language Business Rules**
   presented at the International Conference on Software Engineering Advances 2012
3. **A Model Based Approach for RFID Application Testing**
   presented at the International Conference on Ubiquious Computing and Communications 2013
4. **RFID Systems from a Cyber-Physical Systems Perspective**
   presented at the Workshop on Intelligent Systems and Embedded Systems 2013

Acknowledgements

"All models are wrong, but some models are useful."
George E.P. Box (1979)

I'd like to express my sincere thanks to my first supervisor Prof. Dr. Helge Janicke and my supervisor at the Technische Hochschule Ingolstadt Prof. Dr. Christian Facchi for their trust in me, their patience as well as for the helpful discussions, guidance and the continuous support during my study.

In particular, I want express my special thanks to Raphael Riebl, Christina Obermaier and my brother Tobias Hübner for their support and constructive words during the creation of this thesis.

I would also like to thank my former colleagues, Dr.-Ing. Martin Bornschlegl, Dr. Dennis Böhmländer, Antje Köhler and Markus Meyer for their mental support, pleasant working atmosphere and their creative suggestions.

Last but not the least, I would like to thank my family: especially my mother Ursula Hübner and my sister Regine Hübner for continuous support throughout my life.

Thank you all!

Contents

List of Figures

List of Tables

1 Introduction

This chapter is concerned with the motivation, explores the research gap and presents the research questions derived from it. Subsequently, a novel approach for testing RFID applications is proposed and the objectives of the research are defined. Afterwards, the research strategy and the research method used during the work is described. Finally, a outline of this thesis is given.

1.1 Introduction into the Research Topic

New markets, increased volatility, globally operating market participants, fast-moving sales markets, customer-specific products and demanding manufacturing processes require more flexible and responsive production systems, logistics and employees. At the same time, it is important to maintain the same high level of productivity and quality. In response to these developments, the complexity of industrial processes has increased sharply and continues to increase continuously.

In this context, the question arises as to what the production work of the future will look like?

Current developments, such as the "Industry 4.0", promise new paths to keep up with the new demands. Industry 4.0 is a term that goes back to the German Research Union and a project of the the Federal Government's High-Tech Strategy [4]. At the same time other countries have established similar names, whose goals are largely identical but not always the same (e. g. "Industrial Value-Chain Initiative" in Japan or "Industrie du futur" in France).

The technical background of Industry 4.0 is provided by intelligent and digitally networked systems. Through intelligent systems, processes between machines should be autonomous and self-organized. By networking or connecting them, it is possible to optimize not just one production step but an entire value-added chain and, hence, keep up with the afore mentioned challenges.

Industry 4.0 is based on four pillars:

- **Networking**:
  Machines, devices, sensors and people communicate with one another.
- **Information transparency**:
  Sensor data extend classical information systems to create a virtual image of the real world.

- **Technical assistance**:
  Assistance systems support people with aggregated, visualized and understandable information to help them to make solid decisions and solve problems faster. In addition, people should be physically supported at tiring or dangerous work.
- **Decentralised decisions**:
  Cyber-physical systems (CPS) should make decisions autonomously and only transfer tasks to higher authorities in case of serious problems or disturbances.

However, challenges arise not only in the production processes, but also in the IT systems managing these and in the way data is gathered, since this establishes the basis on which these systems decide. One of the ways to capture data is the automatic identification of objects.

In many areas of today's life automatic identification technology plays an important role. It allows to identify persons or goods and is necessary to ensure quality, security or improve speed and reliability of processes. Especially in traditional domains with a lot of competition, like distribution and logistics, automatic identification systems enable the processes to be more up-to-date, reliable and cost efficient.

The identification systems frequently used in the logistics sector are the well-known barcode systems, although they are limited in their use: One of the biggest issues is the need of a direct line-of-sight to read the data stored on the barcode label attached to the object. Furthermore, the fact that data stored on a barcode cannot be altered any more, prevents agile business processes and "smart" products. Beside, its limited data capacity also environmental influences like physical forces, e.g. scratches or dirt, prevent the correct reading of the stored data. Thus, there is a strong need for a new technology.

RFID [2] was introduced to overcome shortcomings of these prior systems. RFID technology uses radio waves to identify objects. The use of radio waves brings several advantages to the automatic identification process, e.g. a direct line-of-sight is not needed any more. Unlike optical systems (like barcode), radio waves are capable to penetrate materials. This allows for the detection of goods even if they are contained inside a box or a container. Even further, reading over large distances and identification of many items at once (bulk-reading) is possible.

Because of these enormous benefits, it is not surprising that the use of the new technology can be found in many realms of life and in a wide variety of applications. Today, RFID is used to optimize processes, improve traceability, ensure authenticity, enable automatic warehouse management or simplify access control.

With the upcoming demands in the production of the future, RFID is a cornerstone, with it capabilities of identifying goods without human interaction, bulk reading and no need for a line of sight. But also many approaches in the context of Industry 4.0 require autonomy of the systems, autonomous action, knowledge of one's own position and the position of the surrounding objects. The communication between the product (e.g. workpiece) and the production plant is part of this future scenario: the product itself brings its production information in machine-readable form. On the basis of this data, the product's path through the production plant and the individual production steps are controlled. Another example, where it can be used beneficially are "smart refrigerators", which are intelligent systems with embedded computation and RFID readers, capable to order for themselves or at least detect when they are empty and send a message to inform that they need to be refilled. For this reason, hardly any other technology is as suitable as RFID (Radio Frequency Identification) for realising future challenges as outlined above.

Nevertheless, the use of radio technology brings disadvantages as well, one is the sensitivity to environmental influences. This leads to the necessity to have robust systems and testing methods for both physical and software layers, to ensure the quality and reliability of the identification process. However, this data capturing process still remains error prone, even though the basics of RFID technology are available since the early 50's. The technology itself evolved over the years and research efforts increased, but many questions still remain open. Until today, there are only a few publications about testing RFID systems. The available literature mainly focuses evaluating performance aspects of RFID middleware or on testing the physical layer. But RFID applications, which are essential for providing meaning to the gathered RFID data, have not been considered, yet. Therefore, this work tries to solve the question if it is possible to define a methodical approach on testing these applications. Additionally, it composes itself to provide a solid model to describe the environment, improve the efficiency of test case design and test execution for RFID applications and its infrastructure.

1.2 Problem Statement

Although RFID already exists for many years, it is still gaining momentum and strives to compliment or substitute the legacy identification systems in the mid and long run. Today, big companies like Wall Mart, Target, Best Buy, Department of Defence (USA) [5] and Metro AG, Kaufhof, Gerry Webber [6, 7] are already using RFID in their supply chain and many others are currently implementing or evaluating RFID systems for their needs. But *small and medium sized enterprises* (SME) are still struggling to adopt this new technology into their business environment. One of the main reasons for reluctant adoption by SME's is the immense investment necessary, which is significantly determined by the related software and the cost of the RFID tags (item-level) [8].

RFID based systems are mostly highly customized systems, perfectly aligned to fulfil the specific business needs. The software development process is, therefore, a very customized process and repetitive for every RFID project. Sometimes this process includes a variety of sensors and complex design decisions. The customized nature of RFID software raises the need for a specially tailored software test, which itself is a costly process. However, many RFID projects, especially, in small and medium sized businesses, already struggle with the economical value of the RFID solution. This emphasizes the need for cost savings, which might be supported by an efficient test methodology. In addition, one of the main drivers for successful RFID projects are comprehensive system tests [9].

Other obstacles also prevent the technology to be adopted in industry. The following list highlights some of the currently existing issues:

Cost for testing and development - In general the costs for software are distributed over planning, development and testing, while testing takes up to 50% of the overall costs [10]. In accordance to the World Quality Report 2017-2018 [11] a continued demand in finding efficiencies at every level of quality assurance and testing is desperately needed. Especially in regard to the digital innovations like the *Internet of Things* (IOT), the authors predict a increase of testing cost up to 40% in 2019. To lower the effort in developing RFID applications it is important to find a standardised and more sophisticated way of testing, than the manual techniques currently often used in industry. This would allow for reducing the costs and improve the overall development process of RFID applications.

Manual testing - A serious problem of today's practical testing is that most systems are still tested manually [11]. RFID systems are no exception. However, the testers of these systems are usually highly trained people, able to identify problems by using their broad in-depth experience with RFID. Insufficient knowledge transfer capabilities cause enormous efforts for testing, unreproducible tests and exploding costs. Many of today's practical RFID testing is unsystematic, which can lead to an erroneous and costly test process. Despite the costs, it is also too time consuming to repeat tests on that basis. When it comes to automated testing, reproducible tests are not feasible without a virtualized environment within realistic time and cost constraints. Additionally, it is expensive to fund a dedicated physical RFID test environment.

Change of processes - Most RFID systems are composed of RFID readers and an additional layer called RFID middleware. Many merchandise management systems, in which barcodes are used for identification, do not record goods individually but in groups. The electronic product code of common barcode systems, such as the *Global Trade Identification Number* (GTIN) formerly known as *Electronic Article Number* (EAN), consists of a prefix, a check digit at its end, a manufacturer code and one product group identifier. With RFID, goods can be uniquely identified, meaning in addition to the product group identifier it contains a unique serial number for the item [12]. This not only changes the structure of the warehouse database, previously aggregating products of the same type, but also often results in a change of the entire process in the company [5]. So, the introduction of RFID changes the IT systems [13, 12, 14] and in general requires them to be re-tested.

Increased complexity - A reason for the delayed adoption of RFID systems is the increased complexity and change of the underlying business processes. This can be seen as a barrier especially for SME companies. A comparison of RFID systems with *Cyber-Physical System* (CPS) [15] shows, that RFID systems can be treated as a subset of CPS's. Therefore, RFID inherits the same obstacles from the underlying physical processes. E.g. the continuous progression of physical processes, like movements of goods, are computed by computational systems. But the implementations of the underlying business processes are usually not able to handle continuous and truly parallel processes [16, 17]. As a result the complexity of the application is significantly increased by the introduction of RFID systems. The increased complexity can influence software quality aspects negatively. Hence, a test methodology taking the challenges of these systems into account is needed.

Robust and fail-proof applications - RFID technology promises to increase the level of visibility and automation throughout the supply-chain [18], but also implies that automatic identification and tracking needs to be fault tolerant. One of the biggest benefits of using RFID is the reduction of labour costs [7]. However, the failures in autonomous automatic identification systems without human interaction are highly dangerous, because their effects are usually detected late and lead to missing information. E.g. product movements have not been recorded and, therefore, the stock information of the warehouse management system is incorrect. Hence, identification systems increase the demand for robust software applications with high quality.

Variety of RFID applications - The broad spectrum of RFID application areas, software and hardware makes it hard to find a unified methodology for testing. For example, RFID is used with various basis technologies, e.g. *High Frequency* (HF), *Ultra High Frequency* (UHF) and microwave, where each frequency struggles with different physical challenges. A harsh physical environment with lots of interference, significantly influences the read rate of RFID readers. Hence, the input of software applications is influenced and it needs to be robust enough to cope with these unreliable reads of physical processes. Further, the different application classes, e.g. logistics, access control, billing etc. additionally impair the chances to find a model suitable for all these systems. Nonetheless, software tests emphasize different test objectives. This leads to different tests for functional testing, performance testing or robustness testing, to only name a few.

Software and physical challenges - Additionally, technologies based on radio waves - such as RFID - are heavily affected by the physical environment. The propagation of radio signals strongly depends on the physical environment, which, in turn can cause unreliable read rates. Moreover, the environmental effects or the placement of tags on objects can change over time and are hard to predict or even to calculate. Hence, testing the reliability of the involved software systems is important. But there are also challenges for the RFID software in particular. Since RFID readings are continuous (at least can happen at very short intervals), the RFID reader will repeatably report all tags, that it senses at every time epoch. This needs to be addressed by the involved software applications in form of applying filters, aggregation and some times even more sophisticated methods like *Complex Event Processing* (CEP) or *Artificial Intelligence* (AI).

Missing literature on testing RFID applications - The work is diverse and deals with different aspects of RFID systems, e.g. how RFID functionality can be distinguished in different service sets [19], how the management and monitoring of these systems could look like [20], how testing of the air-interface can be done [21], how the interference of radio fields can be tuned [22] or how an integrated development environment could ease application programming [23]. Despite RFID technology being broadly discussed in literature since 2000 and the sheer amount of recent research in the area of RFID, methodical approaches on testing RFID applications have not been considered, yet. The few publications existing on testing RFID [24, 25, 26, 27] focus on performance evaluation, in particular for RFID middleware. Further information on literature and related work can be found in Section 2.3.5. However, common to all the research work is that none of the approaches enable software testing of earlier development stages or on higher levels of RFID applications.

To summarize the challenges, current RFID projects struggle with the following problems:

- methodical approaches to testing RFID applications are still missing
- manual testing is still state of the art and can only be carried out with immense time investment
- tests are not reproducible due to the changing physical environment
- system tests can only be performed after deployment and thus result in a late test phase
- automatic identification technology needs to be fail proof (reliable / robust)

In conclusion, existing software testing techniques, test tools, test processes and test methodologies have to be evaluated in order to determine if their use is adequate for the special needs of RFID software development. A sophisticated environment for testing must be either determined or developed where the application development can take place with dedicated RFID testing. RFID readers need to be virtualized so that the application developer can start the software development before the RFID infrastructure is deployed. Additionally, on larger development teams it is crucial that they do not conflict with their team members, which can be caused by configuring the limited amount of hardware differently.

To address the identified issues, testing is of special interest for RFID systems. But testing increases the development effort and overall costs of the software, contrary to lower the financial barrier. Thus, efficient solutions for testing, especially for RFID systems with high automation potentials, need to be found.

For efficient test automation, a framework must provide three core capabilities: describing sensor input, deriving artificial sensor data, and injecting data into the tested application. Model-based testing promises to lower the expenses through easily comprehensible test descriptions, simplicity in building and modifying the models and in particular though automation of test case generation. A model-based approach also increases the level of abstraction, which leads to incomplete information and makes it hard to automatically extract test cases. Hence, the challenge is to find a well balanced solution addressing the afore mentioned problems.

This work defines a methodical approach on testing RFID systems. Additionally, it provides a solid model to describe the environment, improve the efficiency of test case design and test execution for RFID applications.

1.3 Research Question

The general research question is, how to efficiently test RFID applications in a repeatable and controllable (especially for the environment) manner? In order to answer this question, typical challenges and the consequences thereof need to be considered, e.g. the implications of missed tag reads and which data is suited for testing different scenarios.

The development of a better testing methodology for RFID applications, which addresses the above mentioned issues, raises the question: "How to define a test methodology for RFID software development?". This question can be decomposed further:

- How can RFID applications be tested in general?
- Is there a systematic way of choosing test cases?
- How can RFID applications and their input be described in regard to testing?
- How to simulate the interactions between the actors of an RFID system and the *Software under Test* (SUT) to allow automated test execution?

1.4 Proposal and Objectives

The basic concept to solve the above mentioned questions is a methodology based on a virtual RFID environment that enables developers to create efficient, transparent and reusable automated tests even at very early stages of application development. Furthermore, using virtual RFID devices helps to reproduce tests in a defined environment, saving time and increasing quality of the application.

1.4.1 Proposal

The idea of the novel approach proposed in this thesis is to capture a RFID system's infrastructure and additional attributes of the system in regard to the test objectives in a model. To allow for an easy integration into development processes, it is based on the commonly known *Unified Modelling Language* (UML) [28]. The way the models need to be created is supported by a methodology. For this the requirements of the *Software Under Test* (SUT) are analysed, use case models are created and successively refined with test related information. Afterwards, test cases are generated using these models, enriched with corresponding test data. The resulting tests are used as inputs for a application specific tailored test execution framework and supported through a virtual environment of RFID devices. Finally, the virtual environment is instrumented by the tests generated in this way and the application is evaluated with regard to its correct functionality. Figure 1.1 shows a schematic model of the approach proposed.

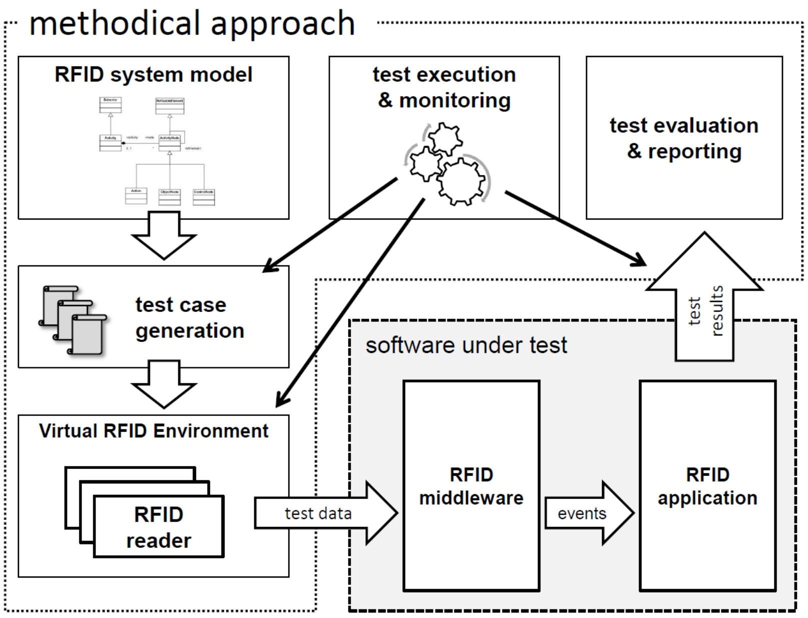


Figure 1.1: Schematic model of the approach

To lower the extra investment for dedicated RFID devices used for testing and enable easily repeatable test environments, this approach uses an open source virtual RFID framework [29], capable to emulate virtual RFID devices. The use of a virtual framework also allows to generate input for the RFID middleware and RFID application in an automated and repeatable manner. The emulation of the RFID devices is also suitable for black-box testing and helps to cope with the different vendor-specific RFID equipment through abstraction.

Unlike other approaches [26, 30, 31], this work focuses on testing functional aspects of RFID applications. It is intended to be used for system testing and only requires the specification as a prerequisite. As a basis it borrows concepts from black-box testing [10, 32], as usual for system level testing, combined with model based testing techniques [33, 34]. It defines three different types of models, which cover the required data for testing. The models themselves are kept simple, because each model only covers a few well-defined aspects. An accurate description of the RFID system under test is achieved through partial linking of these models. This keeps each separate model comprehensible, whilst although allowing to automatically extract test cases for execution. It reduces the efforts for building test cases, as it makes use of UML, which is commonly used in software development and thus does not require additional training. Costs for the test set-up are reduced by using a virtual RFID framework, which allows to execute results of the model extraction. Using models to build the test cases, in combination with instantly available virtual readers, speeds up the test creation and enables faster development and test cycles - even during the early development phases. The flexibility for adjustments of test cases and early testing is an important factor for agile methods especially.

To deal with the challenges discussed in Section 3, different aspects of the approach need to be considered:

Test setup and RFID environment - One approach to testing functional aspects of software is to provide input data and evaluate the resulting actions of the SUT, commonly known as black-box testing. In an RFID system the input data consists of RFID events, which are the result of RFID tags located in the antenna's field of sight. Once RFID events are generated, the events are processed by the middleware and finally handed to the RFID application which executes the business logic.

In today's practical testing the RFID events are produced by persons moving RFID tags through the readers field of sight. The resulting effects on the application are then manually evaluated. This procedure leads to an enormous time effort for the tests, where additional tests significantly increase this effort even more. Additionally, the RFID devices are already cost intensive, and building an additional RFID environment for system testing purposes introduces further costs. With many developers and testers trying to access the same physical RFID devices at the same time, it is almost impossible to restore the same test environment again. This is a result of physical interferences and effects on the air interface (radio) which are hard to measure and non-deterministic.

Model-Based Testing - Model Based testing is a well known approach to systematically identify test cases and can therefore improve the test process. Here models are used to decrease the complexity of test case design and allow a simpler description of the system by introducing a higher level of abstraction. Because of the abstraction, the model covers the system and its specification only partly. Additionally, these models often lack important information, essential to generate executable test cases. Increasing the model's level of detail leads not only to higher costs during the development of the model but also increases its complexity and the risk of introducing errors.

Modelling Language and Domain Model - UML is used in this approach to allow for the use of domain specific terms in the models and as a basis for the models itself. UML also nicely integrates into most software development processes, since it is already used in many disciplines of today's software engineering. Therefore, the effort to set up the models for this test approach is minimized through the already existing knowledge of UML. The created artefacts may also be used in other parts of the development process.

This approach uses three different model types: the Domain Model, the Environmental Model and the Process Models. The Domain Model covers all terms used in the domain, allows to define application specific constrains and contains the relationship between objects. The Environmental Model captures the physical layout and processes and, thus, provides essential information on moving routes and timing behaviour. The Process Models cover the requirements of the software, the business processes surrounding them and are used for mapping the input data and expected outputs of the SUT. These are the most important models of the proposed methodology for testing RFID applications and are used to automatically extract the test cases. To achieve this, each Process model is refined in several steps and then transformed into an intermediate form, representing a abstract test case. Additional parameters from the other models are extracted and used to complete the abstract test cases which are transformed to executable test code eventually. The resulting tests are then executed by the test automation framework, consisting of a virtual RFID environment and application specific drivers. Finally, the test results are evaluated using the expected outputs of the SUT specified in the tests.

1.4.2 Expected Contribution and Goals

The aim of this work is to contribute to quality assurance methods of the SUT and thus, to the reduction of failures in complex, application-centered RFID systems by means of a new, systematic test method - ITERA. RFID systems are strongly influenced by their environments and usually communicate with RFID tags in this environment via error-prone processes. There are currently no specialized systematic approaches for this system type that support a thorough functional test. However, the growing importance and complexity of RFID systems must also be taken into account in the test. The core contribution of this dissertation is threefold: A semantic model which allows to describe the RFID system and the application's requirements, a methodology and guideline for modelling test cases for RFID applications and a test environment for automated test execution.

This thesis has the following goals:

Design of an RFID specific test model - Systematic creation of an system model as a basis for test case modelling, which focuses on the properties of a system, that are relevant for testing and at the same time forms a technology-independent basis for further processing into automated test cases. Beside the systems models (e.g. UML class models, UML behavioural diagrams) an additional meta model to capture the test specific details and its environmental factors is developed. The successively refined systems models combined with the developed meta model build the the basis for the transformation of abstract test cases into concrete test cases and allow for automatic test execution.

Method for generating RFID test cases - For the automatic model-driven generation of test cases from artefacts of the system design, a method for test case generation is developed. Generating test cases from artifacts in the system design requires careful modelling, but in contrast to manual test case creation can ensure that critical components are not neglected.

Method and Environment for automatic execution of RFID based test - A method and accompanying tool support is being developed for the automatic execution of tests and the generation of test reports. The execution of test cases is time-consuming and prone to errors. Automation makes the process reproducible, safe and economical.

1.4.3 Success Criteria

To evaluate the resulting test methodology, the following objectives need to be reached:

- overall the method needs to provide the possibility to model functional tests
- set of approved test methods for functional testing and robustness evaluation
- the resulting model must be able to express all test relevant aspects as system-internal, physical, interaction and contextual information
- the requirement specification must be sufficiently formalised or at least as formal as possible to allow for the derivation of test cases, generation of executable tests which are supported by the test environment
- increasing the efficiency of the test process with a high degree of automation
- to ensure that users can validate the requirements to the greatest extent possible, the resulting model must allow the derivation of user-friendly views.

1.5 Research Strategy and Methodology

In order to answer the formulated research questions and achieve the identified goals, the research has been carried out in five phases:

1. **Review of literature and existing approaches for testing RFID application**
   In a first step, relevant preparatory work is identified on the basis of a literature analysis. In order to gain a basic understanding of today's software development and quality assurance, we first examine the typical software engineering process, followed by software testing techniques used in modern software development. We also deal with the basics of RFID technology, as this is the main focus of this work. Subsequently, we conduct a critical examination and discussion of the relevant work in the field of test case generation and testing of RFID systems. Finally, we will explain what contribution the related work has made to the requirements for testing RFID software.
2. **Analysis of typical RFID Systems**
   Through an analysis of the RFID system and its components challenges are examined. We identify typical sources of error and suggest how these can be taken into account for testing. We also investigate the strong spatial and temporal relation of RFID data and derive a first step into a temporal model that takes this into account. Subsequently, a test environment will be sketched that meets the findings of the analysis.
3. **Development of a methodology and suitable models**
   The focus of this work is on the development of a methodology for creating test models. The method used is based on reference modelling, which later enables the reuse of the

developed model as the basis for a customized model that can be used in different domains and applications. It includes the formal specification of semantics for RFID systems, the creation of models that can depict the environment, the processes and the attributes to be tested. As well as the systematic derivation of test cases from these models and the development of a test environment in order to execute and evaluate the resulting tests.

4. **Identification of use cases and feasibility study of the developed method**
   This involves applying the method to various application scenarios. The goal is to assess the universality and demonstrate that the developed method is flexible and allows a broad spectrum of RFID applications. The selection is based on widely used examples of various commercial applications.

5. **Validation of the methodology**
   The validation shows that the methodology presented here is applicable and assessable with common metrics that correspond to the state of the art for testing software.

1.6 Thesis Outline

The work conducted during this thesis is structured in six chapters:

Literature Review and Related Work on Testing and RFID - An overview about RFID and testing fundamentals is given in Chapter 2, as well as related work in the area of software testing, model based testing and RFID testing.

Challenges of RFID Systems and Testing Requirements Analysis - In Chapter 3 the structure of typical RFID systems, their components and the interactions between the individual parts is explored. Furthermore, the challenges of using RFID and the implications for software testing will be considered. The knowledge gained here will later serve as a basis for creating appropriate tests and for designing robustness tests.

ITERA Test Methodology and Model Based Testing of RFID Applications - A model based specification technique to cover different aspects of RFID software in regard to application testing is introduced in Chapter 4. The formal definition, notation and their semantics are introduced in the first part of this chapter. Therefore, an abstract model that describes RFID system is introduced. The definition of time appropriate for RFID systems is followed by a set-based description of RFID readers and the corresponding RFID events. The same semantics are then used to describe the states of RFID applications.

The second part of this chapter presents a test methodology to generate test cases. For this purpose use cases are identified and analysed. Each use case is refined by activity diagrams, which are used as a basis for systematic test case derivation. The combination the activity diagrams of the use cases, the set-based semantics and the meta model allow to generate executable tests for RFID applications suitable for automated testing. Additionally, the transformation of modelled aspects of the RFID application to tests, fulfilling the defined coverage criteria is shown and considerations for robustness testing are introduced.

ITERA Test Execution Framework - In Chapter 5 a framework to execute the tests extracted from the test model of the RFID use cases is presented. It is centred around a simulation environment with virtual RFID devices building the base of the test harness. Additionally the different parts of the test execution framework are explained.

Evaluation of the ITERA Test Methodology - An evaluation of the proposed solution and methodology is performed in Chapter 6. The validation of the method with regard to its practical benefits is the subject of this chapter. The evaluation is concerned with assessing the methodology on the one hand and the extent to which it meets RFID specific requirements on the other.

Contributions and Future Work - Chapter 7 summarizes the original work contributed to the body of knowledge and suggestions for improvements and future work are given.

2 Literature Review and Related Work on Testing RFID

In this chapter an overview about RFID and testing fundamentals is given, as well as related work in the area of software testing, software modelling, requirement analysis and specification.

Common techniques of software testing have been used and incorporated in this work for RFID application testing. Therefore, the used terms and some principles are explained first (Section 2.1). An overview of the general Model Based Test process, the state of the art and a thematic classification is discussed in Section 2.2. In order to test RFID systems, it is necessary to understand the fundamental structure and the general functionality of RFID systems, the products offered on the market, as well as standards used in practice which is presented in Section 2.3.

Section 2.3.1 highlights the basic components of an RFID system, explains the functionality of RFID middleware and RFID application, and provides a list of today's RFID hard- and software. The components of RFID systems are often using standards for communication, which are presented in 2.3.3. Additionally, an evaluation of the industry standards is conducted and presented. The knowledge of these standards allows to identify starting points for testing and test execution and provides a better understanding about possible error sources. Finally, related work in the area of Testing RFID Systems is presented in Section 2.3.5.

2.1 Testing Fundamentals

According to Myers [10], one of the most important questions in computer science is: "How do you ensure that all of the software you produce does what it was designed to do and, just as important, does not what it isn't supposed to do.". Even though, the first version of his book was published in 1979, testing still plays an important role in any software development process. However, the central difficulty of testing is that it is almost impossible to ensure that there are no errors left. The testing of even simple programs is not an easy task. Exhaustive testing, such as demonstrating that there are no errors present any more by executing it with all possible inputs, is generally impossible. Moreover, exhaustive testing is very costly and therefore only of theoretical importance. As a result, economic considerations become important for testing, as testing will never cover all aspects. However, methodical approaches, which try to structure the test systematic can help to reduce the efforts for testing while still covering relevant parts of the input domain. Furthermore, by test automation also economic aspects of software testing can be addressed.

From a critical point of view, the development of computer software, especially for larger projects, can no longer be captured in its entirety by the human brain. The growing complexity of systems and the inability of the brain to correctly grasp and control this complexity inevitably causes errors. According to this view, a software error is always based on a faulty understanding of program behaviour by the programmer, tester or reviewer: the human interpretation deviates from the actual meaning as calculated by the machine.

The main reason for conducting tests on a software product is to improve confidence in the quality and fulfilment of the system requirements. Errors usually occur during software development. A developer makes mistakes for many reasons. A mistake can be a conceptual error due to misinterpretation, simple distraction and much more. Furthermore, errors can arise at every level of the software development, from requirements engineering up to the implementation and integration of the software application.

2.1.1 Integration into the development process

Testing is an important aspect in all currently accepted software development methodologies (V-Model, Spiral model etc.) in terms of quality assurance and detection of undesirable behaviour in the implemented software or component. Basically, testing can be used for both, validation by users and customers in the context of the evaluation of prototypes, as well as for verification of fulfilment in accordance with a given specification. Testing is therefore closely related to the description of requirements and boundary conditions of the intended future use of the program in the early phases of software development. On this basis, tests allow the observation and analysis of the software's behaviour on planned and unexpected inputs. As described above, tests of software are the de-facto standard in practice to detect unexpected behaviour by comparison with the intended behaviour.

There are two basic problems to overcome when testing software:

- **Variety of Faults**
  Incorrect behaviour can be experienced in different forms that require different tests. A system may not be capable of doing what is expected, e.g. not delivering the expected expenditure or displaying a stock list incorrectly. However, it can also do unexpected things, such as crashing the entire system, e.g. by undefined input or processing unintentional events. Depending on the involved components of a system, different aspects such as data stored in a database, communication with subcomponents, execution of the program logic or the response of a user interface must be checked. However, the diversity of possible faults makes it very difficult to assess the complete behaviour of a software system with an acceptable amount of tests.

- **Complexity of Software**
  Software systems are usually complex so that not every possible execution sequence can be explicitly checked for incorrect behaviour. In order to ensure the best possible coverage of the test space, often metrics are used in static program analysis a type of white-box test, in order to reduce the number of test cases. Within the framework of so-called black-box tests, a series of methods are used to generate typical permitted inputs and particularly error-prone boundary cases as representatives for all possible inputs. However, these methods have to be applied correctly to reach a good level of quality. The tool support for these procedures, such as the generation of typical user interactions when testing requirements, is just as indispensable as a highly automated test execution.

2.1.2 Testing

The activities of the software development process, which focus on ensuring the quality of a software product, are often mistakenly summarized by the term testing. In a broader definition of this term, these include planning activities such as the development of a test strategy and the creation of test plans up to the actual testing, which is the execution of the software under test with different stimuli.

According to the Myers [10], testing is defined as:

> “Testing is the process of executing a program with the intent of finding errors.”

This is similar to the IEEE “Guide to the Software Engineering Body of Knowledge” [35], where testing is defined as:

> “Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems.”

However, since there are various definitions available in literature, in “The IEEE Standard Glossary of Software Engineering Terminology” [36] testing is defined more generally without a explicit focus on finding errors as:

> “The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.”

In contrast to the definition of Myers et al., the standard glossary of the IEEE must be differentiated to the extent that this refers to a value-neutral assessment of the tested system. Myers et al. argue that humans act in a target-oriented manner and that the achievement of the goal is extremely important in the human value model. This effect also occurs when testing software by subconsciously selecting or creating test cases or test data that are unlikely to detect faults in the software. Myers et al. assume a psychological paradox here, because the value-adding activity of software development is the creation of a correct system and not the creation of tests that do not find defects.

Patton [32] uses a similar definition of the term testing and postulates quality criteria for a software tester:

> "The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed."

Targeting early detection of defects is also a more prominent goal here than mere neutral observation and evaluation of the specification conformity of software. But even though that these definitions clarify the term they do not propose concrete tasks or processes how to test the software.

Testing is in most cases aimed at finding faults in a program. Tests can be used to prove that a program for does not work certain inputs under certain boundary conditions, i.e. that it contains an error. Since the input space can be very large even for small programs, the program cannot usually be tested with all possible inputs under all possible conditions for reasons of time and cost. Due to its incomplete character, testing does not prove the correctness of programs. Rather, testing increases confidence in the correctness of the program if no errors are detected during execution of the test cases.

Very well known in this respect is the statement of Edsger Dijkstra in [37]:

> "Program testing can be used to show the presence of bugs, but never to show their absence!"

According to the definitions above, the only purpose of testing is to find errors. If there were no errors, no tests would be necessary. However, due to the complexity of most systems errors are inevitable, because the human brain does not capture the complexity well. By dividing a system into components, this complexity can be reduced to a certain degree that is controllable by humans; however, the complexity problem remains intact during the definition of the relevant test cases, since the system must be considered as a whole. For this reason, the test case selection is one of most important steps for the test and, hence, the main focus of most test methods. In many cases, if not even in most cases, this is a manual activity. With everything that goes along with it, such as possible errors in the test that are caused by missing or wrongly interpreted requirements or by a faulty test case description or an inexperienced tester.

In software development, a distinction is made between different types of software defects [38]:

- A mistake (error) is a mental process that leads to faults.
- A fault (software error, defect) is a deviation between an intended and an actually implemented software product. Every mistake inevitably leads to errors.
- An error (error condition) represents a deviance between the intended and the realized internal state, e. g. an incorrect variable assignment. A fault can lead to an error, but does not have to.
- A failure (failure of the program) is a deviation between the intended and the actual output of the program. An error can lead to a program failure, but does not have to.

Within the scope of this work, the term testing is used as part of the software process, which includes all necessary activities to provide a reproducible validation of the software product's compliance with its specification within the framework of a defined test strategy. The ITERA method, developed in this thesis, aims to improve the quality of RFID applications in different ways.

Inconsistencies between the requirement specification and the test model can be identified and eliminated when the test models are created. This minimizes the risk that mistakes are made during the implementation of the requirements. However, the main focus is on providing testable models, test case generation and the detection of errors by executing the test cases created with this method and thus reducing the probability of undetected failures.

2.1.3 Testing Terminology

To clarify the terms used in this work, the definitions from [36] have been adopted:

test "An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component."

test case "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance."

test bed / test harness "An environment containing the hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test."

test case generator/ test generator "A software tool that accepts as input source code, test criteria, specifications, or data structure definition; uses these inputs to generate test input data; and, sometimes, determines expected results."

test case specification "A document that specifies the test inputs, execution conditions, and predicted results for an item to be tested."

test criteria "The criteria that a system or component must meet in order to pass a given test."

test driver/ test engine "A software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results."

test phase "The period of time in the software life cycle during which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied."

functional testing “Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.”

structural testing “Testing that takes into account the internal mechanism of a system or component. Types include branch testing, path testing, statement testing.”

2.1.4 Software Quality

The overall reason for conducting software tests is to have confidence in the quality and fulfilment of system requirements. According to the ISO25010 [1], there are several quality aspects of software. A distinction can be made between external and internal quality characteristics:

The **external quality** refers to the fulfilment of properties that can be evaluated by any user of the software, such as functionality, usability, reliability, availability.

The **internal quality** is the fulfilment of attributes that can only be evaluated by people who have insight into the source code of the component. Examples of such features are maintainability, reusability or portability.

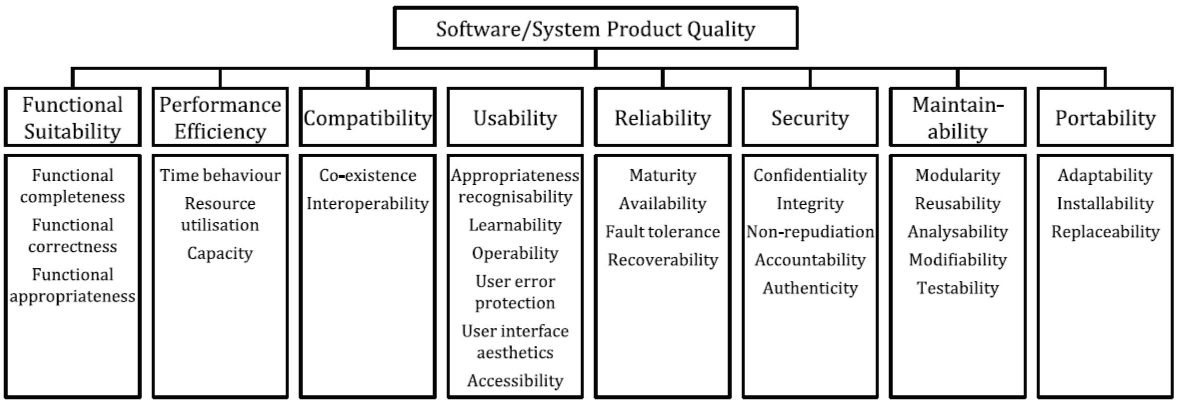


Figure 2.1: ISO/IEC 25010 product quality model (source: [1])

Figure 2.1 shows the quality model of ISO 25010[1] and its classifications of different quality aspects of software. However, the quality attributes are of different importance for different domains; for example, high reliability is particularly important for safety-critical control systems, whereas high availability is likely to have priority for web applications.

Functional test (also requirements-based test, feature test) is the testing of a functional unit against its functional requirements and thus shows the correctness and completeness against the specification. This involves checking the intended behaviour from the specification against the actual behaviour. However, the behaviour of the system should not only be checked in a positive

[1]ISO/IEC 25010:2011

sense, but also the counter test (error test, negative test) should be performed to see whether erroneous scenarios are processed correctly.

A robustness test (also called a negative test, provocation test) is the testing of the system in case of incorrect inputs, which do not correspond to the requirements, in order to check whether the application reacts as intended. This can therefore be seen as an extension of the positive tests. A robustness test is successful when an error message appears or the application responds reasonably to the error.

The procedures considered in this thesis are aimed at checking the quality features of a RFID software system in regard to functionality and robustness. These have been identified as important aspects for these type of applications, since functionality is important regardless of the application domain and robustness plays an important role as the technical challenges for RFID (as shown in Chapter 3) cannot always be solved.

Since errors during development can incur high costs, especially when they are discovered in late stages of software development, two different approaches to assure quality are used in practice. These can be divided into analytical and constructive methods:

- **Constructive quality** assurance tries to *avoid* errors by taking appropriate measures during development. Methods, design principles, formal procedures, development tools and process models are often used for this purpose.
- **Analytical quality** assurance tries to *detect* errors as early as possible and thus evaluate the quality of the artefacts produced. Especially all test activities that assess the achieved quality in comparison to the required quality are used here.

The procedure examined in this thesis refers primarily to analytical measures.

2.1.5 Test Process

A central aspect of software testing is the embedment in a structured procedure when carrying out the tests. The ISTQB[2] offers a test process suitable for the ITERA methodology. The test process is divided into five steps as described in [39]: Test Planning and Control, Test Analysis and Design, Test Implementation and Execution, Test Evaluation and Reporting, Test Closure Activities.

1. **Test Planning and Control**
   Test planning includes the definition of the test strategy and the extent to which the test is to be carried out. In order to determine the scope correctly, it is necessary to consider which software components are to be tested and what the concrete test objectives are. It is recommended to also establish a timetable and determine the resources required to schedule them. Furthermore, test planning includes sett-up of test metrics and definition of input and output criteria.

[2]International Software Testing Qualifications Board - https://www.istqb.org/

Test control deals with the tracking of tests. Test results are measured and analysed taking into account the parameters from test planning. Test progress, test coverage and initial criteria are also monitored and documented. If deviations from the plan are detected, it may also be necessary to adjust the plan. In practice, test control should therefore provide feedback to test planning so that the scenarios that led to the deviation can be taken into account in the future.

2. **Test Analysis and Design**
   The second phase - test analysis and test design - becomes even more concrete. It examines the test basis; the goal is to create a test specification. In order to obtain valuable information for the test, interviews are conducted with the developers, users etc. Furthermore, the testability of the test basis and test objects are evaluated and test conditions are identified. Based on this information, the design of abstract test cases can begin. These test cases can even be prioritized at this stage. The issue of traceability should also not be forgotten, i.e. the reference from test basis to test cases and vice versa. However, not only the test cases are specified in this phase, but also the test environments are described in more detail.

3. **Test Implementation and Execution** In the third phase names “Test Implementation and Execution”, concrete test cases are derived from the test specification, which contains abstract test cases. If required, these can be combined in test suites. Prioritization from the previous phase is largely applied to the test cases and a test execution plan is defined. In order for the tests to be carried out afterwards, the concrete test data must be created and made available.

   Once all test cases have been created, the test execution can start. Regardless of whether the test is carried out manually or automatically, the desired and actual results are compared in both variants. If deviations occur, the causes must be analysed. Regardless of whether there are deviations or not, each test case is recorded in detail so that the test can be reproduced later.

4. **Test Evaluation and Reporting** After the test follows, the evaluation of the initial criteria and the report. The main objective is to assess whether the initial criteria are met and whether further tests have to be carried out. The outcome of this phase is the final test report, which is distributed to the relevant decision-makers.

5. **Test Closure Activities** In the final phase, the test end, mainly finalising work is carried out. The system checks whether all work results have been created and delivered, and whether any open errors and deviations must be converted into change requests. A very important point, which often receives little attention, is the documentation and archiving of test equipment and test environment. These could be important for follow-up projects and maintenance, but also facilitate the later reproduction of errors.

The topics dealt with in this dissertation relate in particular to test specification and test execution. Of course, test planning and evaluation aspects are also taken into account. Requirements for the test specification are worked out in Chapter 3. The subsequent design of the test specification is subject of Chapter 4. Finally, the test execution is described in Chapter 5.

2.1.6 Classifications of Test Methods

There exists numerous ways to classify the available test methods. Depending on literature used, they are for example classified by the granularity of the test object, the sight on the test object (black-box vs. white-box) and the property to be demonstrated by test validation. In terms of the test object views, functional and structural methods can be distinguished, whereby hybrid forms or hybrid tests are also possible. However, they can also be classified by temporal aspects. Real-time tests detect temporal properties of the test object, such as the reaction time to an event or the duration of a particular operation. The concurrency test considers the execution of the test object together with other competing applications or components. The load test examines the test object with regard to extreme quantities of certain data, the stress test with regard to extreme quantities of certain events. With UI-testing, the mapping between internal states of the test object and the user interface is checked. Usability tests address the ergonomic adequacy of the test object.

As depicted, there exists a variety of approaches to testing, therefore, to only those with a strong relationship to this work are further addressed in the following subsections.

Testing Methods

Testing of software is well studied and as it would clearly exceed the limits of this thesis, only a brief overview will be provided. Further details on the techniques can be found in [35, 10, 40, 36].

The broad variety of testing techniques, can be categorized by the type of execution and based on the creation of test cases. The most obvious way to differentiate the available testing strategies is by distinguishing between Dynamic Testing and Static Testing (Static Analysis).

Dynamic Testing - Following the definition in [35] dynamic testing is defined as:

> “This means that we execute the program with specific input values to find failures in its behaviour. In contrast, *static* techniques (e.g. inspections, walkthroughs, and static analysis tools) do not require execution of the program. One of the big advantages of (dynamic) testing is that we are executing the actual program either in its real environment or in an environment with simulated interfaces as close as possible to the real environment. So we are not only testing that the design and code are correct, but we are also testing the compiler, the libraries, the operating system, and network support and so on.”

In dynamic software testing, executable test objects are executed systematically with the aim of proving the correct implementation of the requirements. This reveals the effects of errors and increases confidence in the test object. At the beginning of the test procedure, selected inputs and applicable outputs are determined based on the specification. After the execution of the test, these are then compared with each other. Depending on the test objective, a distinction is made between functional and non-functional tests. Whereby, non-functional tests check quality criteria such as performance or usability.

For the specification of dynamic tests it is necessary to define both pre-conditions and post-conditions in addition to the actual input. These conditions describe the test object's state or states of it's environment. These can also include certain values that are expected in a database or how they have changed afterwards. This way, the tests are reproducible because they can always be carried out under the same conditions. The test cases, especially input and expected output data, are usually derived from the requirement specification.

Static Testing - Static testing or static analysis, in contrast to dynamic testing, is carried out without the execution of the test object. Typical static methods are code reviews or tool-supported formal methods, where the source code of an application is checked against a set of formal rules to identify coding errors. As this type of quality improvement often requires the source code to be available, which is not a prerequisite of the approach proposed in this thesis, it will not be considered any further. However, static approaches can also be applied on other documents, e.g. requirements specifications.

A further distinction is often made in literature (see [10, 41, 32]) between white-box and black-box test procedures.

White-box Testing - White-box testing or logic-driven testing is a testing technique which examines the internal structure of a software under test. The test data is derived from the program logic. One example for white-box testing is a control flow oriented approach. A goal of this technique might be to execute every statement in the program at least once. Often, these goals are defined in terms of coverage criteria. Depending on the chosen criteria, in practice problems arise with loops or through the amount of different conditions.

Black-box Testing - Also referred to as functional testing, data-driven or input-/output-driven testing. Black-box tests, view the software under test as a black-box. The principle of this technique is to neglect the knowledge about the internal structure of the system and find circumstances where the software is not behaving according to the specification. Therefore, the test data used to stimulate the software is derived solely from the specification and not from knowledge about the internals of the program. However, this makes it hard for developers of the software to write the test cases, since they already have a understanding of internal processes. To counter this issue, a principle of Test Driven Development [42] is, that the tests have to be created before the actual development of the software.

Grey-box testing - Grey-box testing or glass-box testing is a combination of both testing techniques explained before. It involves the knowledge about the internal structure of the program to allow a better way of selecting input data. However, the tests are executed like black-box tests. Advantage is, that the tester can choose better test cases since he knows about details of the implementation. This enables to test such things as exception handling, where internal knowledge of the SUT is necessary.

The subject of this work is the (dynamic) test, in which the system to be tested is executed. During test execution, also the SUT is executed and stimulated by inputs, especially through the use of virtual RFID readers, the expected result is then compared through available system interfaces (e.g. accessing a database). Misbehaviour is detected if the observed output does not match the expected output. Further, it uses a black-box approach since the test cases are extracted from the specification, represented by the business processes and additional information on the physical layout and timing. No specific knowledge of the concrete implementation is needed. The models, which are a result from an analysis of the specification, are transformed to executable tests. However, for the selection of meaningful input data which allows for automatic execution of the test cases generated might need some insight of the implemented system in order to evaluate the test result (like a database scheme).

A differentiation with regard to the criterion for selecting test cases is more helpful than the split between the white-box and black-box test: If the selection of test cases is to be based primarily on the required functional characteristics of the system, this represents a function-oriented test. If, on the other hand, it is the goal that the set of test cases, the test suite, is selected in such a way that the internal structure of the test object is covered to a certain degree, it is a structure-oriented test. This thesis focuses on the function-oriented test. It is assumed that the inner structure of the implementation of the test object is not known. With regard to model-based test case generation it should be noted that the structure of the test model is known and this structure is also included in the test case selection. However, this model is an abstract specification of the required functionality of the test object. Furthermore, no connection between the structure of the test model and the inner structure of the test object is assumed. Thus, from the perspective of the test object, it is a function-oriented (or black box) test and not a structure-oriented test.

Test Data Selection - Because of the fact that exhaustive testing is often not feasible in practice, test-data-selection strategies need to be found. This allows to reduce the number of test cases and leads to efficient testing saving costs and time. According to [35] these strategies are called “Input domain-based techniques”.

Different techniques can be used to reduce the amount of test data, which is necessary to gain the maximal revenue out of the test cases. The following list names some of the techniques described in literature [32, 39]:

- equivalence partitioning
- pairwise testing

- boundary-value analysis
- random testing

In the context of this work, however, test-data-selection techniques receive only little attention. The main focus is on determining test cases by structural properties, in particular by analysing the control flow of activity diagrams. The resulting test cases, thus, already limit the input domain as they contain conditions, which are reflected by the generated test paths.

Testing Level and System Test

Tests for software systems can be carried out at different levels. Each level assesses different artefacts of the system to be tested and they are usually roughly aligned to the phases of software development. However, the characteristic feature is primarily the object to be examined during the test procedure. Test stages are usually defined by their proximity to the source code of the system to be tested or to the respective application level. The range of test levels extends from component tests to integration tests, system tests, acceptance tests and regression tests. A test stage is a set of jointly performed test activities that can be assigned to specific abstraction levels of a software system. However, tests at different test levels are not disjoint. A test level is a characterization of an individual test or an execution suite, not a hierarchical delimitation.

- **Module- & Component Test**
  A module or component is tested against the specification of the module or component.
- **Integration Test**
  The modules are combined into composite units and tested against the specification of the technical functional requirements and architecture.
- **System Test**
  The system is tested against the specification of the logical system requirement and architecture.
- **Acceptance Test**
  The system is tested against the user requirements and specifications.

The system test and acceptance test are very similar in their basic structure, but differ in their the observers. The acceptance test is intended to show the customer that the required functions have been implemented. The system test is carried out by the contractor and is meant to demonstrate that the product meets the requirements and is ready for delivery. As a rule, the acceptance test is therefore usually carried out in the actual operating environment. In contrast to this, the system test tries to create a test environment that is as close as possible to the operating environment.

The identification of defects contributes continuously to product quality, which is the reason testing should be carried out at all system levels as early as possible. The system test is therefore embedded between the integration test of a module and the acceptance test. This means that it is a question of viewing at and testing the assembled modules. Accordingly, the system test aims to secure the application before it is integrated into an overall system for the first time. The focus of the proposed test methodology in this thesis is on the system level. For this reason, system level tests will be discusses in more detail.

The purpose of the system test is to test the integrated system to verify that the specified requirements are met by the product. The reason for this is that many functions and system properties only come from the interdependence of all system components and therefore only become verifiable there. Main objective of the system test is to detect errors before economic or physical damage occurs, build up confidence in the system, prove the functionality of the system for certain boundary conditions (requirements) and achieve a specified test coverage. The basis for system tests are all software requirements, specifications and documents that have the entire system as their topic. As the requirements can only be verified on the basis of externally observable behaviour, the SUT is often considered a black box. Therefore, the system's inputs and outputs are mainly used to specify the test cases.

In this thesis the test object is the complete system that is executed in a test environment as close as possible to the actual system environment. The system environment therefore consists of hardware (RFID reader, RFID tag), software (RFID middleware, application software) and the network connecting the individual subsystems. Carrying out RFID tests in the real system environment is a major challenge, as the physical and temporal effects play a major role. Therefore, an approach based on virtual readers that are very similar to real readers was chosen in this thesis. The chosen approach does not correspond to the classical system test, but tries to come as close as possible to a real system environment. When checking the system functions, the configuration settings of the individual components are also taken into account. The system requirements of an RFID application are therefore related to the interaction with reading devices, functional requirements and the states of the data associated with RFID tags. Since RFID systems communicate via error-prone air interfaces, the challenge is to create a suitable test environment.

2.2 Model Based Testing and Test Case Generation

The creation of software tests is one of the more demanding activities of software development. However, testing is not usually an activity that directly contributes to the progress of a software development. In order to reduce the cost of creating software tests and at the same time to increase test quality, technologies for the automatic generation of tests have been researched for a long time. The goal is to either reuse existing artifacts of the system design, the implementation or to generate specific tests from abstract descriptions.

2.2.1 Automatic Test Case Generation

Compared to the actual test, this activity is performed less frequently, but is strongly oriented to the specification of the SUT and requires much higher cognitive performance. The regeneration of test cases is usually necessary whenever the specification of the software system has changed (e.g. adding or removing requirements by the client or iterating the incremental software development) or if defects are observed in the operation of a software whose possible cause has not yet been covered by test cases.

An economic risk of manually creating test cases results from the increasing complexity of the test code with progressive development of a software product. Therefore, the generation of tests can help to control of test complexity and to reduce the effort compared to manual test case creation. To avoid economic damage from defective software, it is necessary to create high quality test cases with a risk adequate test coverage. Due to the high test complexity of software systems, this task is demanding for human actors. For example, functional areas of a software system can be inadvertently overlooked or their significance for the overall qualitative impression of a software system could be misjudged. In both cases, it is to be feared that defects in the test case specification will only be recognised late and a correction of the test case specification will require unscheduled corrective measures.

On the one hand, there is the potential to minimise human error sources by providing technical support for test case generation. Also software tools supporting a human actor in his work in such a way that frequent sources of error (e.g. overlooking requirements due to high software complexity) become less likely. On the other hand, it is also possible to speed up the revision of test cases by drawing human actors' attention to areas of the software, where changes in the specification require adjustments to tests.

Test Coverage Criteria

Testing software systems requires the execution of the SUT under defined conditions and the interpretation of the behaviour of the SUT in order to draw conclusions about the correct implementation of the system specification. Software systems generally implement more than just functionality. In order to fully test a software system, all functional properties must be addressed during testing. However, this is not possible with respect to the problem addressed in this thesis, since only the RFID-specific properties of the application are considered. Nevertheless, the same metrics can be applied to the RFID-specific tests.

Coverage criteria for software tests determine which parts (functions) of a software are addressed by tests. Coverage criteria usually refer to the control flowchart of a software whose nodes represent individual statements of the source code. Coverage criteria have already been defined and examined in the scientific literature (see Patton [32], Myers et al. [10], [39]). These define four different coverage criteria:

When speaking of coverage in the context of activity diagrams, it is typically about the systematic traversal and structural coverage of the directed graph that the activity diagram spans. Thus, the concrete goal is to cover individual (1) nodes (activity or actions), (2) edges (transition or flow edges) or (3) paths. The latter can be restricted to certain subclasses if necessary. E.g. (1) simple paths without inner loops, (2) primary paths (corresponding to simple paths with maximum length) or (3) semantically meaningful predefined paths describing a certain user behaviour, so-called scenarios.

Testing by covering all paths is a practically or even theoretically unsolvable task, as loops can lead to an infinite number of paths. Thus, all practicable methods are based on a limited number of paths. The special constructs of activity diagrams can also prove to be problematic, which enable to model concurrency using fork and join concepts. The resulting non-determinism, in relation to the sequence of events (i. e. interleaving) in the I/O behaviour that can be observed from outside, may lead to a large number of possible constellations that need to be taken into account during testing. Sometimes, certain sequences of events can also be considered equivalent. However, these are not easy to identify without having a corresponding oracle, for example.

Statement coverage testing (C_0 tests) fulfils a coverage criterion that requires each statement contained in the code of the software system to be addressed by at least one test case, i.e. executed as part of the test execution.

The criterion **branch coverage**, on the other hand, requires test cases for each possible alternative sequence of the control flow at each branch. Branch coverage tests subsumes statement coverage because this criterion necessarily reaches all statements of the code. In the literature, branch coverage tests are referred to as C_1 tests. Usually, this criterion is regarded as a minimum test criterion for control flow oriented testing.

The **path coverage** (C_2 tests) criterion requires that all possible paths of the control flow graphs are covered by a test. However, this category is further divided into C_{2a}, C_{2b} and C_{2c} tests. C_{2a} tests require a complete path coverage where all possible paths are tested. It is not feasible in practice to cover C_{2a}, because the control flow graphs may contain cycles that lead to an infinite number of paths. To avoid this, the C_{2b} coverage defines that cycles in the control flow graph are executed at most twice. The C_{2c} test coverage criterion extends the C_{2b} in the sense that here coverage is reached when a specified number of cycle passes are performed.

In contrast to the the previous coverage tests, the **condition coverage** provides a different view by examining the logical structure of conditional decisions at control flow branches. These are often composed of several sub-decisions. In the terms of the C_3 test, these are broken down into atomic decisions and evaluated individually to be true or false. Similar to C_2 tests, a distinction is made between C_{3a} (simple condition coverage), C_{3b} (modified condition coverage) and C_{3c} (multiple condition coverage). Since the individual consideration of the partial conditions often leads to a combinatorial explosion, in practice a variation of C_3b is applied. This variant, called minimal conditional coverage, requires that each of the atomic predicates and the composite condition are evaluated to be true and false.

2.2.2 General Model Based Testing Process

The approach to generate tests from design documents of a software system has been established in science as *Model Driven Testing* (MDT) [48]. The goal is to generate tests from the same system models from which the implementation is derived to maximize congruence between system specification, implementation, and tests. Due to the outstanding position of the *Unified Modelling Notation* (UML), MDT technologies are usually based on UML models.

Hartman et al. [33] explicitly distinguish the term *Model Based Testing* (MBT) from MDT. According to the authors, MDT mainly originated from the *Model Driven Architecture* (MDA) of the *Object Management Group* (OMG) and therefore describes a specific style of the MBT, which differentiates between platform independent and place-specific test models according to the concepts of MDA. In contrast MBT, is also applicable when the tester documents a mental test case design and uses that model for the derivation of test cases.

Although in other literature the distinction between MDT and MBT is not always clear, the difference between the two methods is often due to source of the used models. Generally, MDT uses system models which are created for both the development and testing. In contrast, MBT does not rely on the availability of system models being used for software development. Therefore, MBT is a more general term and, depending on the interpretation, encompasses a whole series of concrete techniques, as can be seen in the reviews by Utting et al. [34], Dias Neto et al. [49] and Shirole et al. [50].

MBT approaches have in common that the tests are not derived from the actual implementation. Instead, tests are generated on the basis of a specially developed test model, which ideally comprises only the information relevant for testing. The overriding goal is always that the (partial) tests can be generated as automatically as possible. This is the main difference to classic black box testing.

"Model-based testing is the automation of the design of black-box tests." [51]

A generic process using MBT techniques is presented by Utting et al. [34]. It is based on the following steps: (1) First, a model is built based on the specification. Then selection criteria are defined to describe the test suite. These criteria are then transformed to the test specification to allow the generation of test cases. (2) Afterwards, the test suite is generated based on the test specification. (3) Finally, the test are run in two stages: (3a) Bridging the gap between abstract inputs and actual implementation using an adaptor or test driver. (3b) A verdict is applied indicating the outcomes: "pass", "fail", "inconclusive".

The generation of the required models in the first step should take place independently of already existing implementation artefacts, so that possible errors can be recognized and no "blind spots" arise. The second step in the process is important for the quality and scope of the generated test quantity, since this requires a selection of the tests to be generated. This assumes that an appropriate selection criterion is available for automation. Selection criteria of this type are usually based on a suitable, possibly problem-specific concept of coverage, which in turn is defined by the used model types (state coverage, transition coverage, def-use coverage).

Often MBT processes can be supported by the fact that (1) implementation details of the system are unknown, since the system does not yet exist or development artefacts are not available, (2) the model results in significant simplifications due to a reduction to the essential system behaviour, (3) many (generated) tests are required, or (4) a suitable model is already available.

Models offer the possibility of consciously allowing non-determinism due to their potentially high abstraction. It is possible to model an unpredictable choice between several alternative executions or unobservable state transitions in the SUT [40]. Thus, design decisions such as the ones that have to be made within the implementation process do not yet have to be anticipated. However, this can also impact the automation potential of model based test generation.

2.2.3 Unified Modelling Notation

The Unified Modelling Language (UML), specified by the OMG[3], is a graphical modelling language for the specification, construction and documentation of software components and other systems. Today, UML is the dominant language for software system modelling. The first contact with UML is often during software engineering or development where diagrams in UML are to be created, understood or evaluated within the a software project. UML primarily determines which terms and relationships between these terms are used to specify so-called models - diagrams in UML usually only show a graphical view of sections of these models. UML also suggests a format in which models and diagrams can be exchanged between tools. UML diagrams can be created in different ways. If the notation of the UML is used as a common language to document the design concept of a software on a whiteboard (or paper) in an analysis team, pencils and paper are sufficient as a tool. In practice, however, CASE[4] tools are often used which support the creation of UML diagrams.

Just as natural languages describe themselves in lexicons or grammars, UML was conceived as a language tool that explains itself with some language components [47].

UML's Profiles language provides a lightweight extension mechanism that can be adapted to specific areas of application. The mechanism is described as "lightweight" because it leaves the meta-model of UML unchanged. This can be a decisive advantage, as tools available on the market for creating and maintaining UML models can often only handle models based on the standardized UML meta-model.

Important diagram types used in this work are the class diagram (Domain Model) and the activity diagram (Process Model). Therefore, a brief overview of the diagram types used is given below:

[3]Object Management Group - http://www.omg.org/
[4]CASE - Computer-adided software engineering

Class Diagram - A Class Diagram is a structure diagram of the UML for the graphical representation (modelling) of classes, interfaces and their relationships. In object orientation, a class is an abstract generic term for describing the common structure and behaviour of objects (classification). It is used to abstract objects. Interacting with other classes, they enable the modelling of a delimited system in object-oriented analysis and design.

Activity Diagram - An Activity Diagram is a behavioural diagram of the UML, that graphically represents the cross-linking of elementary actions and their connections with control and data flows. An activity diagram is usually used to describe the flow of an application case, but it is suitable for modelling all activities within a system. In UML, the semantics of the activity diagrams are still close to those of Petri networks and enable the representation of concurrent systems by integrating additional asynchronous communication mechanisms (signal sending and receiving, exception handling). An activity diagram is usually used to specify a process.

2.2.4 Activity Diagrams as a Basis for Test Case Modelling

UML activity diagrams can be used to capture the system's most important behaviours without being very specific about how they are implemented. They are therefore very suitable for tests at system level. It might be noted that the UML activity diagram is only a semi-formal specification of the system. However, in general it is easier to understand than textual descriptions or formal methods. Therefore, they can be used as an graphical artefact for discussing functionality of the future application with the stakeholders.

Activity diagrams are used to model the work-flow and behaviour of a system. The diagram describes the behaviour as a sequence of activities. The basic elements of an UML activity diagram are activities and transitions. An activity of the system or actuator is represented by a rectangle with round corners. Each activity diagram has a start and end node. The start node is represented as a black circle, indicating the beginning of the sequential activities. The end node represents the end of the process, depicted as a bullseye. Transitions are represented by directional arrows that relate activities to each other and thus represent the sequential progression. Activity diagrams can also describe complex processes, as they allow conditional and parallel activities. Conditions are described by branch and merge nodes, both diamond shapes. Branch nodes have only one incoming and at least two outgoing transitions which are usually protected by guards. Parallel processes are represented by a thick synchronisation bar. A fork has one incoming and at least two outgoing transitions. The end of concurrent activities is a join that consists of at least two incoming and one outgoing transition. Partitions, also known as swimlanes, can combine or respectively group related activities. For the interested reader, a more in depth definition of Activity Diagrams can be found in the UML Specification [47].

The methodology for testing RFID applications examined in this dissertation is based on UML models and can therefore be classified in the context of UML-based MBT approaches.

2.2.5 Related Work on Model Based Testing

Model Based Testing has led many researchers to use UML diagrams such as state-chart diagrams, use-case diagrams, sequence diagrams, etc. to generate test cases. The major advantage of these model-based testing techniques is increased productivity and quality by shifting the testing activities to an earlier stage of the software development process and generating test cases that are independent of any particular implementation of the design.

Approaches to the generation of representative scenarios for testing often use intermediate representations for the activity diagrams to be tested, such as (1) normalized activity diagrams (reduced to certain constructs)[54], (2) special directed graphs [66, 58], (3) Petri-nets [65] or (4) test trees [53, 67, 68].

UML Based Test Generation

Briand et al. [71] investigated a UML-based approach to generate tests from system models. The aim of the approach is to generate functional tests, oracles and so-called test drivers from refined system models, whereby the artefact test driver refers to test execution. Specifically, Briand et al. use activity diagrams with descriptions, sequence diagrams, collaboration diagrams and class diagrams to generate test cases. However, the use of activity diagrams in the authors' approach serves only the purpose to derive superordinate system functions that have to be covered by tests. The activity diagrams only illustrate system functions on a high level of abstraction, but do not provide information on the permitted execution sequences of individual actions. For this reason, the authors also analyse the business logic of the application to determine permitted function sequences based on UML activity diagrams that are assigned to individual user roles in the system. In addition to the use case diagram, domain experts expertise must be consulted.

In the approach examined by Briand et al., test data is defined by annotating individual actions in the activity diagrams with informal data objects. The method studied in this dissertation examines the use of elements from meta models for test case modelling. In contrast to their approach in this work the model is supplemented with test data and context parameters through refinement steps.

Cavarra et al. [72] investigated an approach to automatically generate test cases from UML models. In the studied concept, the authors transform UML models into models that represent test cases and test data. To integrate test-relevant data into UML models, the authors design a UML profile that enhances the meta model of UML. The authors base their approach on the assumption that software quickly achieves complexity that can no longer be fully grasped by human actors. Modelling should help to abstract from the technical details and reduce complexity. Cavarra et al. further argue that complexity reduction is necessary, especially in the field of testing, since the state space of possible combinations of inputs into a software system is rapidly increasing to an unmanageable extent. Furthermore, the authors argue that in principle any modelling language could be used for the test generation process. However, the decisive criterion for acceptance by the user is that no additional effort is required to learn the modelling language. Since UML is a standard in software engineering, Cavarra et al. use it as the basis for their research. However, in contrast to the approach examined in this thesis, state chart diagrams are used by the author to represent the test model.

In 2003, Robinson [73] addressed the challenges and opportunities of model-based test generation. The author bases his argumentation for generating tests from models instead of manual creation on the observation that testers are often less technically oriented than software developers and are involved in the software process relatively late. A lack of technical involvement in the software to be tested and a low relation to the project history reduce the quality of tests. A model-based approach can also help less technically experienced testers to quickly develop an abstract understanding of the SUT and focus test development on key product features. The basic idea here is to understand tests no longer as artifacts developed parallel to the software product, but as executable specifications. The author argues that natural language specification documents are often too imprecise for effective test case creation to ensure adequate test coverage. Robinson also discusses the benefits of model-based test generation in terms of the number of individual test cases that are used as a metric of productivity from a management perspective. For complex software, a large number of tests is quickly created. Changing the specification of the software product may render some or all of these tests obsolete and may require the creation of new tests. Manual creation may require a lot of work to revise existing tests, while the effort to generate tests from a model is reduced to modifying the model.

UML activity diagrams basically describe behaviours, especially processes and procedures. Corresponding descriptions are therefore particularly relevant for testing, since they can serve as a test model on the one hand and as a test object on the other hand. In principle, many papers suggest that activity diagrams are primarily suitable for model-based testing [52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63].

Some of the mentioned approaches differ considerably, e.g. with regard to the supported language features or the UML version. There are also works for direct testing of the behaviour described by the diagrams themselves [64, 65]. Most approaches covered here are very unlikely to be generalised as they are often based on a very specific scenario and thus are not applicable to other situations. However, general ideas or techniques can be used.

Algorithms for test case generation

Not only different intermediate representations are used to represent the test model. Also different algorithms are used to traverse the model in order to extract test cases. Technically, test generation and optimization methods are usually based on:

1. genetic algorithms [69],
2. optimization heuristics, as
    a) ant algorithm [67],

    b) adaptive agents [68],
3. search algorithms [66],
4. model-checking [64, 65],
5. random test generation [70, 55]

The approach chosen in this work also follows the general pattern of model-based testing techniques as defined in Section 2.2.2. For the derivation of test cases, a search algorithm has been chosen. In contrast to genetic algorithms, which can lead to different test cases, a breath first search was used to ensure a deterministic behaviour during test case generation. Additionally, one draw-back of genetic algorithms is that they usually do not scale well with complexity, which can be very challenging in larger activity diagrams. However, all the afore mentioned algorithms can also be used, provided that the specific extensions are preserved in terms of the test execution framework (in particular the virtual RFID environment).

2.2.6 Test Automation

Based on the definition of the IEEE [36], an automation tool refers to a software that expects a preprocessed representation of the specification of the software to be tested as input and - if necessary with manual assistance - generates an assessment of the specification conformity of the software to be tested. Automation can extend to the entire test process or selected parts of the test process.

Depending on the test level, the development process already includes the regular and frequent execution of individual tests. Especially for Test Driven Development models, where tests are implemented before the actual source code, a high number of repetitions of tests can be assumed. The resulting personnel efforts through manual testing can be counteracted by using test automation tools. For this purpose, tools are used that decouple individual quality assurance activities to varying degrees from human resources. This results in a number of advantages that improve the testing process both in terms of quality and economy. Aspects in which the use of test automation is advantageous are (1) speed, (2) efficiency, (3) precision and accuracy and (4) relentlessness (Patton [32]).

The advantage of higher speed at automated test execution stems from the fact, that computers are able to run more tests in a shorter period of time than human operators, do not fatigue and are not dependent on biological factors such as the ability to concentrate. Efficiency benefits result from the fact that a developer or tester's manpower is fully utilized during manual testing and is not available for other activities. The manpower released in this way can then be used for other activities, e.g. further software development or creation of additional test cases. Furthermore, machines are superior to human beings in terms of precision and accuracy, as they do not fatigue even in long lasting, monotonous activities. Errors due to lack of concentration, as can be expected from repeated execution of many tests, can be ruled out in the use of test automation. With Relentlessness, Patton describes the property of a machine-based tool to be able to reliably perform tasks of any size, regardless of the expected duration.

Ramler and Wolfmaier [43] dealt with the question which circumstances tests are recommended to be automated. The authors critically discuss the validity of the intuitive assumption that in a software project as many as possible tests should be automated. This contrasts sharply with Bertolino's [44] dream of 100% test automation in a future perspective of testing. As a particular risk to the cost-effectiveness of test automation, Ramler and Wolfmeier identify that costs for the automation of tests will no longer be amortized if the tested functionality is removed from the software product late in the project. Bach [45] also provides a critical contribution, according to which manual testing is superior to automatic testing, especially because of the high costs arising when creating automated tests. Furthermore, Bach argues that both processes typically also address different defects. However, Kasurinen et al. [46] investigated projects in 31 software companies with respect to the use and usefulness of test automation tools. They conclude that test automation is generally suitable for reducing the use of resources for quality assurance activities. In particular, repetitive tasks have been identified as suitable targets for automation to focus rare resources, such as cognitive performance of developers and testers, on more value-added activities.

In this dissertation, iteratively manually refined UML activity diagrams are used as input for an automated test execution framework. Automated parts of the test process are the generation of test cases and their execution while simulating certain aspects of the execution context. Irrespective of the unresolved question whether the increased effort justifies test automation, other factors must also be taken into account. Unreliable data acquisition of RFID systems, which are often due to physical effects, make it difficult to repeat tests under the same conditions. However, the test results should be reproducible and deterministic. This can only be achieved if environmental conditions can be fully controlled. The test automation in this work includes a controllable environment and allows to simulate RFID reading events that are difficult or impossible to generate manually. Thus, test automation in the context of ITERA should not only be seen as pure automation, but also as necessary means to reproduce repeatable tests.

2.3 RFID Fundamentals

In this section an insight into RFID technology is given and the basic principles of the technology are explained. First, the basic concepts of RFID systems are introduced. Then, an overview of the infrastructure and the nature of typical RFID systems and their components is depicted. Additionally, a non exhaustive list of hardware and software in the area of RFID is provided. This covers the most well known vendors and tries to find out which components are used in industry. Finally, the different standards and their adoption in industrial solutions is investigated. The understanding of the acceptance of standards in industry, as well as the components and infrastructure, clarifies how the different components interact and can be used meet the needs of practitioners and to design a test methodology for testing RFID systems.

2.3.1 RFID Technology

RFID is an enabling technology which allows for the automatic collection of data (e.g. identification, location, transaction, and time) quickly and easily without human intervention. Electronic data labels, comprised of integrated circuits and coupling units, are used to store the data. Contrary to similar smart-card systems, where the information is transmitted through a physical connection, RFID uses radio waves instead. The energy needed for the transmission is drawn from the magnetic or electromagnetic field of a interrogator. However, since the basics of RFID technology exist since the 1960's [74, 75] the technological concepts behind RFID are not new any more. Most of them have been adopted from radar and radio technology [2].

Figure 2.2 presents the components of an RFID System and their interactions. All of these systems always consist of at least two components, a RFID reader and a RFID tag. Both are coupled and use a communication channel to exchange data, beside the transfer of power.

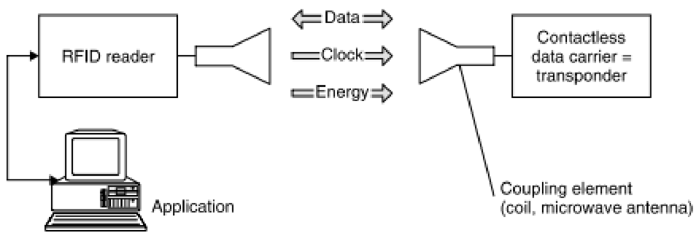

RFID reader
Data
Clock
Energy
Contactless data carrier = transponder
Application
Coupling element (coil, microwave antenna)

Figure 2.2: Typical RFID System Architecture (source: [2])

Typically, the RFID reader is comprised by a radio module, a control unit and a coupling part. Additionally, it offers different interfaces for communication with traditional computational systems. These are often Ethernet, RS232 or other proprietary interfaces, allowing the RFID application to receive the gathered data or to control the device.

The RFID tag usually consists only of a coupling unit and a microchip. Most RFID tags do not have an own power supply. Therefore, they are not capable of transmitting data on their own and stay passive. At least, as long as they are outside of the emitted field of a reader. When a tag is moved into the electronic or magnetic field of a RFID reader, the tag draws power and activates the microchip. However, also a few active RFID tags with a power supply, usually realised by a battery, can be found today.

RFID can be divided into various categories. The most important factors to distinguish between RFID systems are presented in the following list:

- **frequency - LF, HF, UHF, μF**
  The range, where RFID tags can still be read strongly depends on the used frequency.

But each frequency has down- and up-sides, determined by the physical interference with materials in the close surroundings.

- **power supply - active, semi-active, passive**
  Passive tags have no own power source. Therefore, they depend on the electromagnetic or electric field of the corresponding reader to power them. A semi-active tag uses the power of a battery to keep its memory. The active tag does not depend on the reader as a power source.

- **memory - 1-bit, multiple-bit**
  Depending on the memory, the transponder can also be used to store additional information. In some cases the size of the memory is just 1-bit, where either a tag is in the field of the reader or not. As they do not contain a unique identifier it is not possible distinguish between different tags. 1-bit transponders are mainly used in anti-theft solutions, currently used by the majority clothing stores.

- **coupling - magnetic, electromagnetic**
  The maximum range in which the detection of a tag is possible not only depends on the applied frequency, but also on the used coupling technology. As the range of magnetic fields is limited by its field strength, which decreases inversely to the square of the distance, magnetic coupling is only used in close-range scenarios.

Parts of the following subsection have already been published in [29, 120].

2.3.2 RFID System Structure

A RFID system is composed of one or many RFID readers which are connected to a RFID middleware. Figure 2.3 shows a schematic RFID system and the interaction between the components. These RFID readers/devices communicate over radio waves with RFID tags, which are attached to objects the RFID application is interested in. RFID tags are read by the antenna of the reader and the tag's identifiers are reported to the middleware. The RFID middleware filters redundant data, applies logical evaluations and generates business events for the RFID application. Additionally, the middleware is providing unified and transparent access to the collected data for the RFID application. The RFID application, which usually implements the main part of the business logic, interacts with the user and serves as an endpoint for the data.

RFID Reader

An RFID system consists at least of one transponder with a unique identifier (commonly known as a RFID tag) which is attached to or contained within an object, and a device for reading it. The information stored on the tag is continuously read through a RFID reader (or interrogator) equipped with an antenna. The coupling is realized by magnetic alternating fields generated by the reader in a small range or by radio waves for increased ranges. Often coupling is not

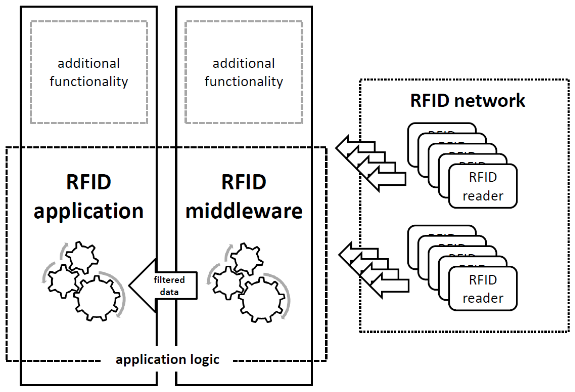
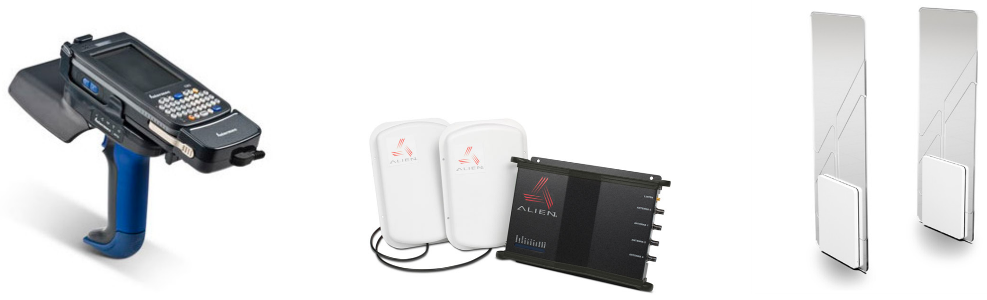


Figure 2.3: Schematic RFID system and interaction between components

only used to transfer data but also supplies the transponder with energy. Active transponders, in contrast to the passive transponders, with their own power supply are used to achieve even longer ranges, but are associated with higher costs and increased maintenance for the battery (charging, replacing). The link between RFID tags and RFID readers is often referred to as "Air-Interface".

The RFID readers are typically connected to the actual application via a so-called RFID middleware. Reasons for this three-tiered architecture are, that typical business applications often struggle with real-time processing and the immense amount of data, which can be generated through a RFID system. Furthermore, the middleware also provides a unified and device independent access to the gathered data. In principle, RFID readers can be operated independently from the typical RFID infrastructure, however, in industrial practice the benefits of RFID often arise through the use of multiple readers.



(a) Intermec IPS30 (b) Alien 9800 (c) Feig Crystal Gate

Figure 2.4: Variations of RFID devices: (a) Mobile Reader (b) Fixed Reader (c) RFID Portal

In practice, various types of devices such as mobile readers (see Figure 2.4a), handheld systems and fixed readers (see Figure 2.4b) are used as gates (see Figure 2.4c) or stand-alone reading terminals. Although there are different categories of readers, all RFID readers have in common that they can sense tags in the field of view. The field of sight of RFID readers is often described as an "interrogation zone" or "read zone".

An interrogation zone consists of the RFID reader, antennas, cables, peripherals and the environment in which the device is installed. The reliability of an interrogation zone is determined by its environment. In some cases, unintentional readings may occur or the reading process is blocked at all. For example, the presence of multiple objects often leads to reflections, absorption or interference with the original signal. In order to eliminate or overcome the environmental and business process obstacles and limitations the hardware and its software (firmware) must be correctly selected and configured.

Peripheral devices can also be connected to the readers. These devices include feedback systems like light stacks or warning horns, providing information about the system's state, or sensors like light barriers to switch the interrogation zone on and off. Peripherals are usually connected via the General Purpose Input/Output Ports (GPIO ports) of the readers. In addition, they also feature interfaces with enabling direct or indirect network communication to alert back-end systems of detected or tracked objects or goods.

According to Floerkemeier et al. [76] there is a wide range of different functionalities provided by RFID readers. However, it depends on the computing resources or the presence of embedded operating systems which functionality can be realized. Therefore, it is possible to differentiate between readers that only provide a basic set of operations and those that are able to execute custom code.

Generally speaking, depending on their capabilities, the readers are called “intelligent” or “dumb”. Intelligent interrogators have enough computing power to run software that filters, aggregates, and analyses data to transform it into meaningful events for the back-end system. In contrast, a dumb interrogator is only able to read tag data and report detected events to the back-end system. Especially the later type is heavily dependent on a middleware product that performs filtering and aggregation functions and converts the read data into a meaningful format that serves as input for the back-end system. However, most of today's readers are intelligent interrogators.

The reading devices often feature a textual interface (Telnet, SSH) or graphical user interface (GUI) via which they can be administered and configured. Typically, access is provided via a web interface or a simple configuration program. The configuration settings usually differ from manufacturer to manufacturer. In the following, some of these configuration parameters are considered as exemplary settings:

Output power - The transmission power setting of the reader is important for reading tags. It is crucial to adjust the power level of each RFID reader to avoid overlapping of adjacent interrogation zones and decrease interference. The power levels are usually given in watts (W), milliwatts (mW), decibel (dB) or percentage (the maximum permissible power varies greatly in the various regions of the world). In some devices, the power for writing tags can also be adjusted independently of the transmission power, since writing operations usually require more energy.

Antenna settings - Often antennas can be grouped together. This allows multiple antennas to act as a single unit. In some cases, it is also possible to set an antenna sequence that allows several antennas to be operated in a certain order without interfering with the signal of other antennas.

Filtering - This functionality allows to communicate only with a subset of the tag population. For example, if there are pallets loaded with several products, a dock door interrogator can filter out all products and only report the pallet.

Polling - Specifies how often the reader scans for tags, either continuously, at pre-set intervals or as needed. The scanning of tags on demand is usually triggered by connected sensors (e.g. light barriers, etc.).

Autonomous and Interactive Mode - This setting specifies how to retrieve read events. In autonomous mode (sometimes called push mode), the events are passed directly to the software systems above. Often a repetition count is also configured to specify how often new tags are searched for in the reading zone before a reading event is generated. In autonomous mode (pull mode), the reader is queried for new reading events by the subsequent systems.

Read Modes - Some RFID readers sometimes come with pre-set read modes. Although the manufacturers use different terms to refer to these modes, they are all basically identical for all interrogators and can be distinguished into two modes: the conveyor mode and the inventory mode.

Conveyor Mode: The conveyor mode is used in situations with high tag mobility and, therefore, short visibility time spans. To cope with this short visibility, the interrogator reads at fast intervals with a short repetitions.

Inventory Mode: The inventory mode is a transition mode, which means the interrogator watches a number of tags for a period of time. Each loss of already watched tags or gain of a new tag will be reported to the back-end system.

RFID Middleware

Typical RFID systems found in today's industry are not only build on a single RFID reader. Instead, they consist of many RFID readers, to obtain the most benefit from the automatic identification technology. However, a larger number of readers also increases the complexity and needs improved maintainability. Nowadays, middlewares are used in many areas to address complexity and allow for a standardised way of accessing the devices. Therefore, the concept of an additional computational system between the RFID devices and the application was introduced in RFID as well.

In a survey of RFID middlewares [77] by Forrester Resarch the **RFID Middleware** is defined as:

> "Platforms for managing RFID data and routing it between tag readers or other auto identification devices and enterprise systems."

A middleware is an application which connects different layers of software in a way that the complexity and infrastructure is hidden. RFID Middleware are also commonly called "Edge Server" because of the placement at the edges of a RFID network. It is the part of the infrastructure enabling the RFID application to communicate with the RFID readers without explicitly knowing the protocol of the different hardware vendors.

In the context of this thesis the middleware can be seen as a central part of the application logic, thus, it needs to be addressed during testing as well. This is achieved by integrating the middleware into the test bed. According to the test execution framework presented in Chapter 5, the real RFID readers are replaced by virtual readers. This preserves the architecture of the RFID system and the middleware becomes an integral part of the evaluation.

The tasks of RFID middleware are manifold. According to [77] the middleware offers many different services. Many functions are available across the various vendors. However, as for the reader, it depends on the solution provider which specific functionality is available. The following list offers an overview of the common characteristics:

Data Management - A problematic aspect of RFID systems is the significantly increased data volume. RFID systems generate considerably more data than conventional identification systems, e.g. barcodes. For barcode systems, for example, a barcode is only scanned if someone triggers the scanning process. The increased quantity of reading events in a RFID environment is a result of the near real-time monitoring of goods movements. RFID readers are continuously reading the tags in their interrogation zone several times during one second. Another reason for the large data volume is the number of tags in typical RFID systems. RFID makes it possible to uniquely identify each object, also known as item-level tagging. For example, whole pallets with up to hundreds of products are scanned at once. In order to cope with the huge amount of data, almost all middlewares support data capture, allow to filter and aggregate the events captured from RFID readers. For example, companies like Wal Mart are already recording a large volume of data with conventional technology. After the planned implementation of RFID, an additional data volume of several terabytes per day is expected. To cope with such high data volumes, the necessary hardware must first be provided. However, due to ever more powerful computers and abstraction of events on every layer of an RFID system, storing data is no longer regarded to be a problem.

RFID Reader & Application Integration - The tag events collected by the middleware must be forwarded to the backend systems so that the events can be processed. Especially in scenarios where several applications need to be informed about events, it is not practical and too complicated to have a separate connection to each reader for each application. In addition, some RFID readers use different protocols that are manufacturer-specific. Application and Reader Integration provides, therefore, a unique interface and enables data to be exchanged between the participating RFID readers, applications.

Reader and Device Management - Since RFID is widely used without human interaction or supervision, the continuous and correct operation of the devices must be ensured. Therefore, modern RFID middlewares monitor and observe the readers to detect malfunctions or defects.

Process Management and Application Development - Since there is no typical business process for RFID, each RFID application has an individual aspect. Middleware adjustments are therefore an important feature. In many cases, the data generated by the RFID system is related to certain business events. These events must be transmitted to the back-end systems so that the event can be handled. A good example is the arrival of new supplies. The deliveries are delivered to the warehouse, which is equipped with RFID readers. The reader identifies the new goods and generates a report stating that the new products are immediately available in the warehouse. In addition, it is very useful for the development process if the way how data is captured can easily be changed. Thus, most RFID middlewares allow for the free design, especially of data capturing mechanisms, to cope with various business processes.

RFID Applications

The use of RFID promises to solve many issues of today's identification systems. Furthermore, improvements and more efficient processes are realisable through the use of it. Therefore, it is not surprising that RFID is used in various application areas and in a variety of businesses. The examples given here allow gaining a better understanding of the different applications a test methodology might need to be suitable for. Additionally, it can be used as a basis for a classification of the applications and provides a reference point which test aspects need to be considered.

Ticketing - Payment with smart cards using magnetic strips is state of the art in public transport. Almost every big city use these systems today. However, because of the physical contact between the smart card and the reading device the wear of the smart cards is enormous. To solve this RFID is already used as a substitute. Skiing areas and public transport are using RFID to replace printed tickets and allow the reuse of tickets. In most cases a plastic card with an embedded RFID tag is read at the turnstiles by a RFID reader to grand access to customers. The Oyster card in London's public transport system is one example. Especially for skiing the contact-less long range access control offers additional convenience for the customers, since it is not necessary to swipe the card anymore.

Payment - Credit cards or debit cards with embedded RFID circuits are becoming more popular. With the embedded RFID tag it is possible to use the credit card without sliding the card through a magnetic reader. But also cafeterias, for example at Universities, are using RFID systems for payment. An advantage of this system is to reduce the physical wear of the magnetic sliding cards.

Access control in cars - Car manufacturers have added security and convenience into automobiles by using RFID technology for anti-theft immobilizers and passive-entry systems. Already, almost every remote control in new cars is equipped with a RFID chip to the check the integrity and allows access prior to start of the engine.

Inventory management & supply chain management - Smart shelves with embedded RFID readers are used to locate goods. This gives a real-time view of the interior of a warehouse or libraries. "Out of Stock" situations can be minimized or avoided. Additionally, WalMart, Target, BestBuy, Metro and other retailers have discovered that RFID technology is helping to manage the supply chain and can keep inventories at the optimal level, limit shoplifting, and speed customers through check-out lines. RFID tagged goods can be identified and automatically checked at dock doors. It is also possible to scan the content of a pallet or box without even opening it. The automation of the scan progress furthermore reduces human failures and makes the process more secure.

Logistics - In this area RFID mainly used for tracking products along the distribution chains. Beside other benefits, monitoring of real-time flow of goods is possible, allowing the customers to track the status of the delivery. It also decreases the management effort of returns.

Production management - In times of globalisation and increased competition just-in-time manufacturing gains more attention. The RFID technology helps to set more insight on the current status of production steps on demand. But, also customization in the production is an popular use case for RFID. Recently, cyber-physical systems additionally push the use of RFID in this area.

Maintenance and repair management - Particularly, the repair management and maintenance is a time consuming task. Since, RFID tags are writeable and offer to store additional data. This can be used to monitor the wear level, but, also expensive utilities can be tracked.

Trade and retail - To improve the availability of products and in consequence increase the revenue of retail, detailed knowledge of the current products in the shelf's and sales rooms is essential. RFID allows providing this information and, therefore, plays an important role in the trade and retail sector.

Health care & pharmaceutical industry - Correct medication is a crucial procedure which allows no failures. RFID plays an important role to ensue quality and lowers the risk of errors. It is self-explanary that the unique identification of patients is a perfect fit of this technology.

Augmented reality - To identify the surroundings RFID tags are placed on objects to provide additional information. For example this is used in museums to improve guided tours with audio systems.

These application areas of RFID already demonstrate that RFID is used in a broad variety of applications. However, the sheer amount of application areas makes it hard to find a unified test methodology, suitable for all areas. Nevertheless, the successful application of a test approach is significantly determined by the abstraction level. Hence, one goal of the proposal should take this into account and provide a sufficient way to describe the attributes of RFID systems.

In this thesis the various application areas are addressed by the use of UML activity diagrams, which are widely used and already allow modelling processes in a variety of use cases. Additionally, an extendable and flexible test execution framework is proposed in order to allow the use of various sensors and different interfaces of the application under test.

RFID Hardware and Software vendors

As shown in the previous section (see 2.3.2), the usage of RFID is very versatile. In order to develop a test methodology, it is important to know the components used in today's industrial applications, since they are part of the test environment. Additionally, the impact and practical value of a test approach depends on the adoption of standards, protocols and functionality employed by the different solutions. Therefore, a brief overview of state-of-the-art RFID system components is given in the following paragraphs. It is divided into hardware and software components of well known vendors. The software suppliers are split into different categories, depending on the coverage degree of their products. Even though, that the results of this investigation are contemporary because of the high fluctuation of vendors in the market, it allows to determine common features and it is beneficial for the practical use of the proposal. Nevertheless, it is infeasible to cover the whole market and every RFID solution available, therefore, only UHF RFID which is said to have the highest impact, is considered here.

Hardware Vendors - Like every market today, the market for RFID is also very diverse. Hence, it is not surprising that there is a wide variety of vendors providing *Electronic Product Code Class-1 Generation-2 UHF Air Interface Protocol* (GEN2) (see Section 2.3.3) compatible RFID reader devices. To gain better understanding of the readers used in industry, and, thus also to understand the components under test, the following list shows a non-exhaustive overview of the most well-known vendors:

- ALIEN Technology (http://www.alientechnology.com)
- AWID (http://www.awid.com/)
- Deister (http://deister.com)
- Intermec Solutions (http://www.intermec.de/)
- Feig (http://www.feig.de/)
- Impinj (http://www.impinj.com/)
- Ident-Pro (http://www.identpro.de/)
- Motorola (http://www.motorola.com/)
- Siemens (hhtp://siemens.com)
- Sirit (http://www.sirit.com/)

All listed vendors offer UHF Gen2 compatible RFID readers, however, the used communication interfaces strongly depend on the vendor. Currently, there are no regulations on how the data has to be exchanged on the physical layer between the reader and the middleware. So, different interfaces can be found. Most common on is the Ethernet Interface for communication. But there are also solutions with a RS-232 Interface or others like RS-422. However, each RFID device is also using a protocol for communication and many vendors have their own proprietary protocols for the readers. This becomes challenging in regard to application integration and for testing, since the test would then also become very specific. For example ALIEN technologies use a "telnet-style" protocol. But there also exist standards in that area. *Low Level Reader Protocol* (LLRP) is the successor of the not very successful *Reader Protocol* (RP), both defined by EPC Global (more detail on LLRP can be found in Section 2.3.3). Generally, two types of protocols can be distinguished:

- Text-encoded protocols
- Byte-encoded protocols

Each protocol encoding has advantages and disadvantages. It depends mainly on the use of the reader. E.g. if a human is interacting with a reader it is much more straight forward to use a text-encoded protocol than a bit-encoded protocol. On the other hand, a bit wise interpreted protocol can save bandwidth. Particularly, when it comes down to a basic challenge of RFID - the huge amount of data generated by the RFID readers - it might be necessary to preserve network capacities, which is easier achievable when the readers utilise a bit-based encoding.

Software Vendors - The amount of application areas already suggests that there are numerous vendors in the area of RFID software as well. According to Forrester Research [77], different categories of software vendors can be identified:

- Platform providers
- RFID pure plays
- Integration specialists
- Supply chain application vendors

Platform Providers - Platform Providers often base their product on bigger platforms, mainly an application server where the business applications run in any-way. The RFID middleware can be seen as an extension to this platform.

- Oracle - RFID and Sensor-Based Services
- Microsoft - BizTalk RFID
- IBM - WebSphere Sensor Events

RFID pure plays - RFID pure plays are usually companies which make all their business with RFID products. In that category often a new and highly customized RFID middleware is found. The Software is small and lean. The main use is to completely fulfil the needs of a distinct RFID use case. Additionally, the companies often have been funded because of RFID business idea.

- GlobeRanger - iMotion Edge Platform
- Reva Systems - RFID Network Infrastructure
- Sybase - RFID Anywhere - archived!
- TrueVUE^{TM} RFID Platform
- S3EDGE - Spotlight RFID Server
- OMNITROL - Edge Application and Sensor Engine (EASE)
- OATSystems - Tool zum Designen von RFID Prozessen
- RFID Innovations (Graz) - RFID Framework SmartID
- RF-iT solutions (Graz) - You-R OPEN Middleware

RFID pure plays with open source models - Universities and open-source communities, sometimes with associated companies, also provide implementations of RFID middleware solutions. The products are available under different open-source software licenses. However, the business is often based on support only and since the source code of the software is available it is listed separately.

- CUHK - EPCGlobal RFID Middleware
- logicAlloy - ALE Server RFID Compliance
- Fosstrak - ALE Server RFID Compliance
- Trancends - Rifidi Edge Server Middleware

RFID application vendors - In this category applies for vendors specialized on RFID business. Their solutions are highly customized and fitted for customers' needs. They have a standard framework which is then customized for the special use case. The product is sometimes also offered as a branch solution (e.g. "Slap-and-Ship")

- Seeburger AG - RFID Slap & Ship Solution
- Manhattan Associates
- Siemens

In summary, an overview of the current RFID market was provided. The overview is contemporary, since many companies in this area disappear and new ones appear frequently. Nevertheless, the knowledge gained here provides a solid basis for the investigation of the common protocols used by RFID readers, for understanding the acceptance of standards and for finding common functionalities between the solutions. Further, it is an important aspect of the test approach, because this is the only way to put practical benefits in the foreground and to find suitable input generators for the test environment.

2.3.3 RFID Standards

In this section the RFID standards defined by EPCglobal are presented. There exists a framework defined by EPCglobal, providing an overview of the standards. It shows how the standards interact with each other and defines different layers allowing to group the standards according to the structure of RFID systems. This framework and the different layers are explained in the following paragraphs. Starting from the lowest level, the interfaces between RFID tags and RFID readers, and ending with the application level. Finally, a evaluation of the adoption rate on standards used in industry is presented.

EPCglobal Architecture Framework - The non-profit organization EPCglobal defined a so called Architecture Framework [13], shown in Figure 2.5 to provide an overview of the standards. It frames all standards defined by EPC Global and describes where these standards are applied. Moreover, the architecture also defines different layers and sorts the standards into these groups.

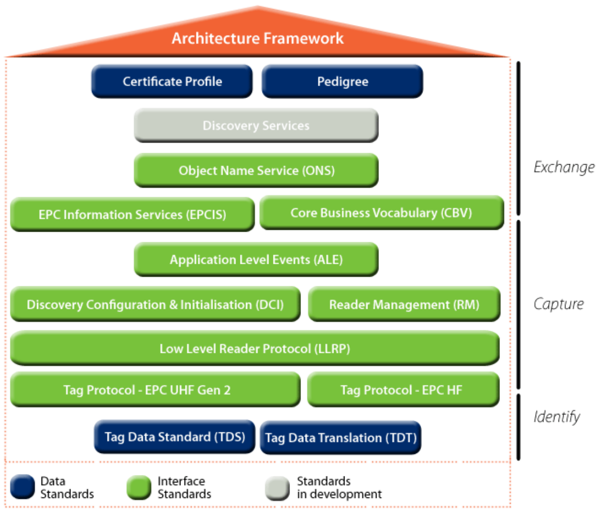

Architecture Framework
Certificate Profile
Pedigree
Discovery Services
Object Name Service (ONS)
EPC Information Services (EPCIS)
Core Business Vocabulary (CBV)
Application Level Events (ALE)
Discovery Configuration & Initialisation (DCI)
Reader Management (RM)
Low Level Reader Protocol (LLRP)
Tag Protocol - EPC UHF Gen 2
Tag Protocol - EPC HF
Tag Data Standard (TDS)
Tag Data Translation (TDT)
Exchange
Capture
Identify
Data Standards
Interface Standards
Standards in development

Figure 2.5: EPCglobal Architecture Overview (source: EPCglobal[5])

The Architecture Framework itself is divided into three layers namely Exchange, Capture and Identify. The Exchange layer is the highest and covers all interaction between Auto-ID Infrastructure systems of different participants. The Capture layer covers the interactions between the back-end systems and the RFID devices or other sources of automatic identification. In the Identify layer the air interfaces and over the air communication is defined.

The information on the standards has been taken from EPCglobal[5]:

Tag Data Standard - The *Tag Data Standard* (TDS) [78] is part of the Identify layer. It focuses on the RFID tags and defines the *Electronic Product Code* (EPC). The EPC is the unique identifier used to identify objects with attached RFID tags. It is similar to the *European Article Number* (EAN) which was super-seeded by the EPC/*Global Trade Identification Number* (GTIN) as result of globalisation. The EPC is defined as a unique identifier for all physical objects, whatever they may be. The allocation of numbers is managed by GS1 which operates several sub companies in almost every country of the world. Additionally to the EPC itself, it also defines the user-memory contents of Gen 2 compatible RFID tags.

In more detail, the TDS covers two broad areas [78]:

- The specification of the EPC, including its representation at various levels of the EPCglobal Architecture Framework and its correspondence to GS1 keys and other existing codes.
- The specification of data that is stored on Gen 2 RFID tags, including the EPC, "user memory" data, control information, and tag manufacture information.

The EPC is divided in several subgroups, each used for a specific purpose and class of objects. The GTIN used for identification of trade items, the *Global Location Number* (GLN) describes locations, the *Serial Shipping Container Code* (SSCC) is typically used for pallets and containers or other logistics load units, the *Global Returnable Asset Identifier* (GRAI) for returnable and reusable assets, the *Global Individual Asset Identifier* (GIAI) for fixed assets, the *Global Service Relation Number* (GSRN) for service relations and the *Global Document Type Identifier* (GDTI) for documents.

Global Tag Data Translation - The current version of the Tag Data Translation [79], which is part of the Identify layer, is 1.6. A machine-readable version of the EPC Tag Data Standard and rules for formatting and translation of EPC identifiers is covered by EPC Tag Data Translation standard. It can be used for either validating EPC formats or translating between the different levels of representation in a consistent way.

EPC Class-1 HF RFID Air Interface Protocol - This standard [80] defines the physical and logical requirements for a passive-backscatter, Interrogator-talks-first, radio-frequency identification system operating at 13.56 MHz. It is divided in three areas: (1) Physical interactions, i.e. the signalling layer of the communication link, between interrogators and tags; (2) interrogator and tag operating procedures and commands and (3) the collision arbitration scheme used to identify a specific tag in a multiple-tag environment.

EPC Class-1 Generation-2 UHF RFID Air Interface Protocol - The "Tag Protocol – UHF Class 1 Gen 2" [81], or GEN2 for short, is the specification which defines the physical and

[5]EPCglobal http://www.gs1.org/epcglobal

logical requirements for passive-backscatter RFID communication in the 860-960 MHz frequency range. It describes the air-interface used by RFID readers and tags for interaction. Moreover, it also defines the data on an RFID tag, the used encodings, anti-collision and so on. Most international and national regulations for RFID technology are based on this standard defined by EPC Global. In Europe, the Tag Protocol was approved as ISO/IEC 18000-6C.

Low Level Reader Protocol - LLRP [82] is part of the Identify Layer and the successor of the Reader Protocol (RP) to define a unified protocol RFID reader vendors should use to ease the communication and make RFID readers more exchangeable. The aim of the LLRP is to enable communication between an RFID reader and an RFID tag (GEN2) and between an RFID reader and RFID applications. In addition, RFID readers can also be controlled and configured using this protocol.

Discovery, Configuration and Initialization Protocol - As part of the Identify layer the *Discovery, Configuration and Initialisation* (DCI) [83] specifies an interface between RFID readers, access controllers and the network on which they operate. The purpose is to simplify maintainance and deployment of a RFID readers. It is used to initialize the reader, configure the network settings and to exchange configuration information.

Reader Management - The *Reader Management Protocol* (RM) [84] is a standard to identify common functionalities and define a protocol to change settings and configure reader devices. It also describes some management functionalities readers should provide. It extends the Reader Protocol which was a first draft to find a standard for reader to application interaction.

Application Level Events - *Application Level Events* (ALE) [85] specifies an interface through which clients may obtain filtered, consolidated EPC data. The data sources can be an RFID middleware or RFID “smart” readers but also other sources like bar-code and legacy identification systems. The use of this standard provides independence between the infrastructure components that acquire the raw EPC data, the architectural components that aggregate & filter the data, and the applications that use the data.

EPC Information Services - The goal of *EPC Information System* (EPCIS) [86] to provide a common standard across enterprises to share collected EPC data and the associated business events (context). It specifies interfaces for recording and querying so-called EPCIS events and enables participants to significantly increase transparency and control over their respective processes.

The EPCIS Data Model includes detailed information of product movements, e.g. the EPC ID, Event Time, Business Step, Disposition, Read Point, Business Location, and Business Transaction. The standard explicitly allows an to extend of the contained information by the participants without revising the specification itself. This allows users to supply additional information like temperature or expiration date, etc.

2.3.4 Rifidi - Virtual RFID Device Simulator

The Rifidi Toolkit enables the virtualisation of RFID readers of various vendors for flexible and improved testing of RFID applications. Utilizing virtual RFID readers in this context means less physical readers need to be available for testing. Therefore, it not only allows for the automatic execution of tests but also lowers the barrier to enter the RFID world by saving resources and time. The Rifidi Toolkit is available since 2006 as open source software on SourceForge.net [129] and already established on the market as a valuable resource for many users, which can be seen on the number of downloads (over 50,000 since 2006 [130]).

Parts of the following subsection have already been published in [29, 120].

Rifidi Toolkit Overview

The Rifidi Toolkit consists of three software tools; namely the *Emulator*, the *Designer* and the *TagStreamer*[3]. All previously named tools are based on the *Emulator Engine*, the central part of the Rifidi tool-suite which is capable of emulating virtual RFID devices.

Emulator - The Rifidi Emulator is a graphical interface for controlling and interacting with virtual readers. It allows to emulate readers and tags as well as read and write events. Additionally to this, it provides access to the virtual readers like their physical counterparts.

Designer - The Rifidi Designer is a tool to build custom 3D production environments, that can be used for visually simulating the RFID data flow. It is also based on the *Emulator Engine* and allows the emulation of RFID readers and the interaction with them. It is not maintained anymore but can still be accessed on the SourceForge.net website.

Tag Streamer - The Rifidi Tag Streamer is a performance testing tool that allows to generate large numbers of virtual readers and tags to evaluate non-functional requirements of the RFID system.

The Rifidi Emulator Engine

The *Emulator Engine* is the core part of the Rifidi Toolkit. It is responsible for managing and controlling the emulation of the RFID devices. One Instance of the *Emulator Engine* can control multiple virtual readers of various vendors at the same time. All parts of the Rifidi Toolkit are implemented in Java and use different technologies around the Eclipse Framework, e.g Equinox, JFace and SWT.

The basic tasks of the *Emulator Engine* are:

- the communication between RFID reader and client
- execute commands issued to the virtual reader

- management of the antennas and the related field of sight
- to control the reader specific components, like signals on the *general purpose input/output* ports (GPIO ports)

Because it uses a Service Oriented Architecture approach, the functionality of the *Emulator Engine* is distributed in two core services. These services can be obtained through the *ServiceRegistry* and used to manage virtual readers, virtual tags and keep track of the tags in the Rifidi Environment.

ReaderManager - This service is responsible to manage the devices and provides of the following actions:

- create a virtual reader
- delete a virtual reader
- start a virtual reader (power on)
- stop a virtual reader (power off)
- add a virtual tag to a reader's field of sight
- remove a virtual tag from a reader's field of sight
- get a list of virtual tags currently in the readers field of sight

IRifidiTagService - This service is used to create and handle virtual tags. It consists of the following functionality:

- create a virtual tag
- delete a virtual tag
- track virtual tag data changes

All existing Rifidi tools are based on the *Emulator Engine* and allow an easy, intuitive and graphical access to the presented functionality. However, the Rifidi Toolkit can also be used as a basis for other tools, which rely on the emulation of virtual RFID devices, e.g. test data generators. But to use the capabilities of the tool-suite for improved testing of RFID applications a deeper insight on the framework is necessary.

Emulator Engine Architecture

This section gives more insight on the architecture of the Rifidi tool-suite and describes how to use this architectural model as a basis for virtual readers, which can be used for test automation of the ITERA methodology. It concludes with an overview of the command flow through the *Emulator Engine.*

A concrete implementation of a virtual reader is called *reader module.* To seamlessly integrate into the framework each *reader module* implements interfaces of the components, presented below.

The architecture of the *Emulator Engine* is based on the structure of a physical RFID reading device. The idea was to develop the virtual readers similar to their physical counterparts and therefore to use the same logical components. Figure 2.6 gives an overview of the different

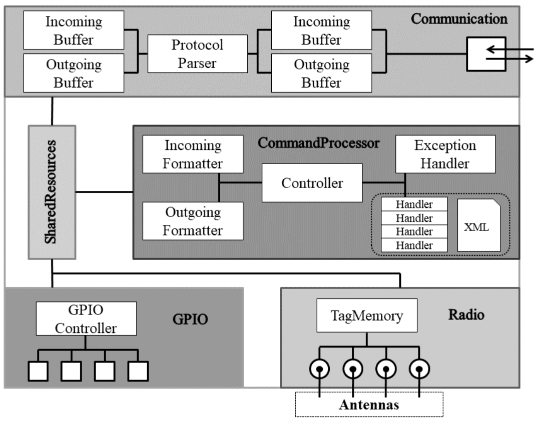


Figure 2.6: Emulator Engine Architecture Overview [3]

components of the *Emulator Engine*. Each box represents a logical counterpart of a physical reader and can be seen as a generally valid abstract component of all virtual readers.

Emulator Engine Components The components of the *Emulator Engine* are Communication, Command Processor, Radio, General Purpose Input/Output and Shared Resources. The different tasks and functionalities of the virtual reader's components are explained in the following:

Communication - Even though most modern RFID devices use an Internet Protocol (IP) based protocol for communication, there are still some vendors using other interfaces like serial link, e.g. RS232, or other proprietary ones. To map these diverse ways of communication to the framework a generic communication part for each virtual reader needs to be considered. In the Rifidi environment this is realized through the communication component. It enables receiving and sending of messages as well as forwarding the encoded messages to or respectively from the command processor. The communication is divided into two parts:

Protocol Parser: The protocol parser converts the messages received in binary format to a message the command handler can deal with. This can be either a reader specific message format or a plain Java object. The *Protocol Parser* is also used this way when it converts a message from the command handler to a binary message, which can be sent through the reader specific communication. With respect to IP-based communication, especially regarding fragmented packets, the parser is also responsible to determine if a message was received completely.

Buffer: The buffer is used to store messages and enable asynchronous communication from the communication channels. There is one buffer for incoming and outgoing messages.

Command Processor - The interaction with common RFID readers is based on command/response protocols. Subsequently the issued commands need to be parsed, executed and finally responded. In the Rifidi environment the command processor can be seen as the central processing unit and is used for this purpose. For each command it invokes the appropriate methods of the virtual reader. A list of commands and the corresponding actions, implementing the functionality of the command, are defined in a XML file called `reader.xml`. The Rifidi framework utilizes *Java Reflections*[131] to execute the classes and methods specified in this XML file. The command processing layer is consisting of three parts:

Formatter: As for the buffers there are formatters for each direction of communication. The incoming formatter is used to strip and decompose the commands. Usually the first parts of a command determine the command type, all additional information are arguments and parameters. The outgoing formatter assembles outgoing messages for further processing in the communication layer.

Controller: The controller, also called command handler, is the central processing unit for commands. Each command issued to the virtual reader is executed here. The *HandlerMethods* are invoked through reflection.

Exception Handler: The exception handler is a special *HandlerMethod*, which takes care of unknown, undefined or misspelled commands. Usually error descriptions are send back to provide feedback.

Radio - The radio component represents the air interaction capabilities of the virtual RFID device and allows to interact with virtual tags. Via the Rifidi Emulator a user can control when a tag is added to an antenna or when a tag is removed. Additionally to the user interaction, the reader can also, depending on the reader capabilities, write or read the tag. Furthermore it is vendor specific how "tag events" are stored and handled in a reader, therefore each virtual reader has its own radio component and memory structure. As an example, the *Alien 9800*[6] RFID reader allows either to list all read events since the last poll or to list just the currently available tags in the field of sight.

Tag Buffer: The tag buffer implements the memory structure and the over-the-air interaction capabilities of the virtual reader. The Tag Buffer is associated to the virtual antennas of a reader. The antennas represent the interfaces for the *Emulator Engine* to simulate RFID tag operation events. It has to be distinguished between read and write operations. Reading tags means a tag's information was read by the reader and writing tags means the tag was modified during time the tag was available in the field of sight. An Event is either a tag appears on the antenna or disappears. In more detail, this means a list of virtual tags is either added to the Antenna or removed from the antenna.

General Purpose Input/Output - Some RFID devices allow additional sensors to be connected to it. This is as previously mentioned, widely known as *General Purpose Input/Output*

[6]Alien Technology Corporation (`www.alientechnology.com`)

(GPIO) and realised as small electrical connectors on the RFID reader. The presence of GPIO ports is also reader specific and the virtual functionality is provided by this part. Currently only the Low Level Reader Protocol (LLRP) Reader and the Alien 9800 Reader leverage this functionality in the Rifidi Framework, yet.

Shared Resources - The shared resources are used as a “housekeeping” component for each virtual reader. It is a central instance holding together the different components and allowing to transfer data objects from the above described layers in this section.

Overview of the Command Flow

Concluding with the different components of an virtual reader, Figure 2.7 gives an overview of the command flow through the different parts of the *Emulator Engine*. It shows a schematic view of the data exchange and interactions of the different components. The dotted lines around the components indicate implementation specific parts, which can be different in each virtual *reader module*. All incoming commands are received by the *IncomingBuffer*, encoded in a binary

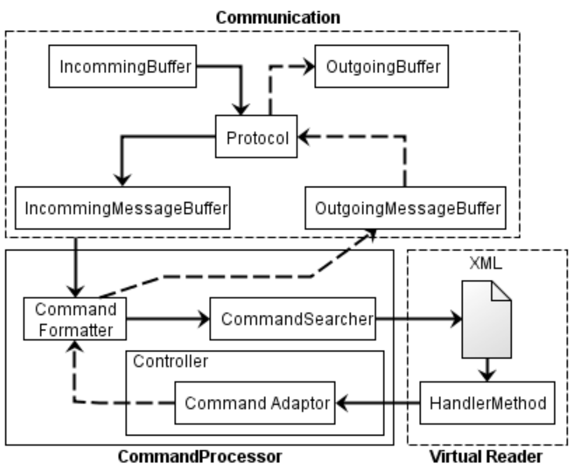


Figure 2.7: Rifidi Command Flow Overview [3]

format. Once all bytes of the message are read, it is forwarded to the *Protocol*. The *Protocol* decodes the binary messages to a processable format and hands it over to the *IncomingMessageBuffer*. The *IncomingMessageBuffer* stores the Message until the *CommandProcessor* is available for processing. The *CommandProcessor*, consisting of the *CommandFormatter*, the *CommandSearcher* and *CommandAdapter*, is parsing the message and looking up the corresponding *HandlerMethod*. Finally, the located method is executed and the intended actions are performed. If there is a response or a result of the previously executed command the data flows the way reverse. Following the dotted arrows, the reply first goes through the *CommandFormatter* again. Afterwards it is given to the *OutgoingMessageBuffer*, then encoded by the Protocol and finally sent to the *OutgoingBuffer*.

2.3.5 Related Work on RFID

This section presents the results of the conducted literature review. Literature which can be divided into three categories: test data generators, performance evaluation and cyber physical systems. First, an overview of the literature on test data generators is given. These can be used to provide input data for the SUT and build the basis for a virtual test environment as envisioned in this work. Then literature on performance evaluation of RFID is considered with speared focus on publications is on performance of RFID middleware. Evaluations on RFID applications, however, on which this work is focused on, can not be found in literature. Finally, an overview of Cyber-Physical Systems is presented. RFID is often used as one of the technologies in the context of CPS, especially with regard to "Industry 4.0". However, it remains unclear if RFID systems are CPS. Therefore, the definitions found in literature are considered and a comparison of the components of both systems is given.

Test Data Generators for RFID

In order to allow testing, especially for black-box tests, the input data for the SUT needs to be generated to stimulate the software. The results of the processing of this data then need to be compared to the expectations, allowing an evaluation of the test. The test data generators used for testing RFID applications can be can be divided into virtual and hardware emulators.

Virtual RFID Toolkits

Beside the Rifidi Toolkit [29] (see also 2.3.4) chosen in this approach, other frameworks for virtual RFID device emulators can be found in literature. The concept behind emulation of real existing devices is not new. It is a common approach to virtualize these to achieve various goals. When it comes to testing, the most prominent reason for virtualization is the control over an environment, which can be influenced as needed by the tester. Therefore, these frameworks are mainly used as input generators for the test execution.

A definition of the terms used in the following subsection is given below:

emulator "A device, computer program, or system that accepts the same inputs and produces the same outputs as a given system." [36]

simulator "A device, computer program, or system that behaves or operates like a given system when provided a set of controlled inputs." [36]

Currently, there are two implementations of virtual RFID emulators mentioned in scientific publications, beside Rifidi. Unfortunately, neither the *Virtual Test Toolkit* from the Pusan National University nor the *RFID Performance Test Tool* from the Feng Chia University have been available for download and comparison. Hence, the further investigations on those are only based on information which could be obtained through publications and not from concrete experiments.

Virtual Test Toolkit - [89], [24], [90], [91] describe a virtual reader emulator to evaluate a RFID middleware mainly with the focus on performance. The toolkit is divided into three parts; the *Virtual Application Emulator*, the *Virtual Device Emulator* and the *Toolkit Operator*. The *Virtual Device Emulator* is comparable to the Rifidi Toolkit. It is capable to emulate one or more virtual RFID readers and virtual RFID tags. The virtual readers can interact with virtual tags, which are following the EPCglobal TDS. A *Toolkit Operator* controls the virtual environment and is the central part of the virtual test toolkit. The *Virtual Application Emulator* is connected to the RFID middleware and acts as a RFID application. The purpose of the *Virtual Application Emulator* is to collect information regarding the performance of the middleware under test for evaluation. The toolkit was first published in 2009 [24] and follows similar objectives as the Rifidi toolkit and the RFID Performance Test Tool. It differs to the Rifidi toolkit mainly in the emulation's level of detail. For example, a fine grained access to the virtual readers including configuring the output format of tag events seems not to be possible. Nevertheless, it can be used as a basis for testing, but it is not as flexible as the Rifidi Toolkit.

RFID Performance Test Tool - Jongyoung and Naesoo [92] introduce the design and implementation of a *Performance Test Tool* for RFID middleware, consisting of a test data generator and a result data estimator. The test data generator supports different tag data standards and various reader protocols, e.g. from EPC Global, Alien Technology and Motorola former Matrics. The data estimation part implements the ALE specification and connects to the RFID middleware, which is the *Software Under Test* (SUT), as a virtual application. The reader emulator, which is part of the test generation tool, is controlled via a graphical user interface and allows to control the amount of tags, that will be generated. In addition to the amount of tag events the simulation is supposed to generate, it allows to define a pattern for the encoding of the generated tag data, as well as a timing interval for the events. Once the specified test data is generated, it is accessible for the RFID middleware through the "result data transmission"-part. In summary, the introduced tool is similar to the concept of the Rifidi TagStreamer. It is a performance test tool to exercise RFID middleware and allows the emulation of vendor specific reader protocols. But, contrast to the TagStreamer, it does not allow fine grained access or interaction with the virtual readers. Nevertheless, the advantage of the *Performance Test Tool* is the result estimation part, which is used the evaluate the performance expectations.

Rifidi Toolkit - The Rifidi Toolkit (see Section 2.3.4) is similar to the other tool-kits listed before. However, it strives to be a emulator, contrary to a plain input generator. Nonetheless, it can be used to generate RFID events and thus provide stimuli to the SUT. The virtual RFID devices provided by this tool behave like their real counterparts, which is achieved through a full protocol emulation. The principles and concept of Rifidi has been discussed in [29]. But, there are further publications in literature available using this toolkit:

Palazzi and Ceriali [93] provided a critical investigation of the capabilities of the Rifidi Toolkit regarding RFID system testing. For this investigation they use the toolkit in a case study to demonstrate the current potentials. Siror et al. [94] use Rifidi as a basis to evaluate the usability of an an automatic evaluation of a customs verification process. In both cases the Rifidi framework was successfully used to simulate the RFID environment and therefore supported the fast and easy demonstration of the case studies. Mueller et al. [95] compare different test data generators and conclude, that the Rifidi Toolkit is an specialized data generator which can generate RFID events by emulating RFID readers.

Rifidi allows for the emulation RFID tags and enables to control the environmental conditions of the environment as needed for the test methodology for RFID applications in this thesis. Furthermore, it fully emulates the RFID reader, which allows a bi-directional communication between the RFID middleware and the RFID reader, contrary to simple input generators as chosen in other approaches. Therefore, and because of its availability as open-source software, it was chosen as a basis for the virtual test environment for the proposed methodology.

Hardware Emulators

Beside software emulators, there are also physical emulators available [21, 96, 97]. The hardware emulators are mainly used to simulate RFID readers, in order to test new air protocols, modifications to existing protocols and new command sets. However, some of the devices can even simulate RFID tags [98]. For example, CISC Semiconductors offers the *RFID Tag Emulator*[99], a mobile device capable of emulating multiple RFID tags. This tag simulator can be used to analyse and test RFID reader performance with certain tag populations. But similar to real RFID readers, these simulators have to cope with limitations regarding high investment costs compared to virtual readers and are affected by physical environmental conditions. Hence, beside economic considerations a full control of the environment in order to allow for reproducible test is not possible.

Performance Evaluation

There are several publications focusing on the evaluation of RFID middleware performance. Additionally to the design for a performance test tool, Lee et. al. [92] define parameters for workload, such as numbers of clients and readers, and determine estimations to evaluate the performance of a RFID middleware. The test set-up is based on a number of virtual RFID readers and simulated application clients. During the execution of the performance test, the number of virtual RFID readers and applications is increased to generate workload for the middleware. The middleware's workload and response times are monitored during the test execution and finally evaluated.

A similar approach was chosen by Zhang et al. [91], which is also based on virtual RFID readers and virtual application clients, but the used evaluation monitor, includes different test objective evaluators. Following objectives are distinguished: robustness, abstraction, performance and reliability. However, in contrast to the approach chosen in this thesis the application's functionality has not been considered.

The tool introduced by Tai Hyun et. al. [100] is not focusing on RFID middleware, but rather on the performance evaluation of *EPC Information Service* (EPCIS). The framework is divided in two parts: data generation and performance test. The *Data Generation* tool uses an *Event Generator* which is used to generate the test data and either store it in a file or a EPCIS database. The *Performance Test* tool generates queries to the EPCIS and monitors the response times.

Contrary to this work, the above mentioned literature on performance evaluation of RFID middleware focuses on different test objectives like: response time, bottleneck identification and throughput.

Simulation of RFID Supply Chains

There are various publications centring around the simulation of RFID supply chain activities. Mueller et. al [27] provide realistic test data for a pharmaceutical supply chain, which was generated using a simulation. The approach is based on the EPC network infrastructure and a tool called Rockwell Arena for the simulation of the RFID supply chain.

Wang et. al [101] also propose a simulation of RFID supply chains based on EPCIS, which is presented as a case study of a frozen food chain. PLANT Simulation is used to describe the logistic processes and generates EPCIS-Events based on XML bindings. EPCIS-Events are transmitted to the EPCIS database in real time to support testing of applications based on the EPCIS.

The above mentioned literature on test data generation, based on simulation of RFID supply chains, underlines the necessity to test RFID software, but all approaches focus on EPCIS systems and do not consider testing functional aspects of RFID applications as does the approach presented in this work.

Testing of RFID Middleware

The most advanced and prospective approach in regard to testing RFID applications is presented by different authors of the *Pusan National University* (PNU) [90, 30, 102, 31]. First, a virtual RFID framework is presented, which is similar to the open source software Rifidi. This framework was enhanced later on and used as a data generator for RFID middleware testing. Additionally, a monitoring tool was added to allow the evaluation of performance impacts on the middleware. Finally, the attributes of RFID data, like movement patterns and unreliable data gathering, have been introduced. The movement pattern used is derived from business processes and based on petri-nets. It represents the paths the RFID tags can move along. It is essential to distinguish between *Semantic Valid Data* (SVD) and *Semantic Invalid Data* (SID), where SID has no semantic meaning encapsulated in the data itself. SID is generated through random RFID events and can be used for performance evaluations, where it is not necessary that the tag's data follow a special numbering scheme. Contrary to SVD, which is generated from business processes and represents RFID events, where the data of the tags include a semantic meaning, which is needed to trigger the intended functionality. Therefore, SVD can be used for functional testing. Nevertheless, an evaluation of the RFID middleware, in regard to functional aspects was not considered yet. Hence, regardless of the advancements achieved for testing RFID middleware solutions, also RFID applications can not sufficiently be tested with this test set-up either.

Cyber Physical Systems

Parts of the following subsection have already been published in [15].

Recently, *Cyber-Physical Systems* gained immense attention in industry as being a technology enabling a new industrial revolution. Current focus of research in CPS is leading to an increasing number of scientific approaches and methodologies for the specification of these systems. In the current literature, however, no clear and consensual definition of what a CPS is has been agreed upon. *Radio Frequency Identification* (RFID) systems are often considered a subset of CPS [103, 16, 17], but a clear analysis in regard to common principles, characteristics and the topology of CPS and RFID systems has not been conducted, yet. This arises the question: Can a RFID system be a Cyber-Physical System? Are all RFID systems always CPS's or not? And as a consequence, can the techniques used in CPS also be used for RFID systems?

A large number of definitions for cyber-physical systems can be found [103, 16, 17, 104, 105, 106, 107, 108]. In the following the various definitions are revisited, compared, and combined into a single definition based on the common characteristics for which the main components of CPS are identified and presented in a schematic overview.

Lee [103, 16, 17] defines CPS as follows:

> "Cyber-physical systems are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa."

With a focus on the dynamic aspects of CPS and emphasizing the physical behaviour Lee et.al. [104], postulates CPS furthermore as:

> "Cyber-Physical Systems integrate the dynamics of the physical processes with those of the software and communication, providing abstractions and modelling, design, and analysis techniques for the integrated whole."

Additional to the previous definition the aspect of a detailed software design is moved in the focus. This is an in interesting point for RFID systems as well. The approach in this work is based on models trying to bridge the gap between the physical processes and the software.

Rajkumar et al. [105] chose a control-theory oriented definition:

> "Such systems that bridge the cyber-world of computing and communications with the physical world are referred to as cyber-physical systems. Cyber-physical systems (CPS) are physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communications core."

In this definition the cooperation of CPS through communication is a major concern.

Tan et al. [106] state that even if a cohesive definition of CPS is missing, all above mentioned criteria are valid and define CPS as:

> "Although the question of "What is a CPS?" remains open, widely recognized and accepted attributes of a CPS include timelines, distributed, reliable, fault-tolerance, security, scalability and autonomous. [...] Cyber-Physical Systems are a next generation network connected collection of loosely coupled distributed cyber systems and physical systems monitored/controlled by user defined semantics laws."

This definition emphasis on distributed systems, especially on specific algorithms, which are designed for distributed systems. However, this aspect is also included in the definition of Lee [103, 16, 17].

Similar to Raijkumar's work [105], a more control-theory oriented approach has been chosen by Talcott et al. [107], specifying CPS as:

> "Cyber-physical systems (CPS) integrate computing and communication with monitoring and/or control of entities in the physical world. Sensing and manipulation of the physical world occurs locally, system behaviour emerges as a result of communication and other forms of interaction."

Park et al. [108] state that CPS can be seen as an evolution of embedded systems:

> "Recently, the convergence of cyber and physical spaces has further transformed traditional embedded systems into cyber-physical systems (CPS), which are characterized by tight integration and coordination between computation and physical processes by means of networking. In CPS, various embedded devices with computational components are networked to monitor, sense and actuate physical elements in the real world."

This definition defines the properties of CPS very good. However, it is based on the definition of embedded systems.

To summarize these definitions, a combination of Lee [103] and Tan [106] has been chosen: A CPS is defined as a highly networked system of systems with a tight integration of physical processes and computational parts. Main cause for increased complexity of CPS is additional functionality, due to the fact that the used techniques are for distributed applications.

The main challenges in the area of CPS are to find accurate models, that are capable to describe the physical processes in addition to the computational model, as well as choosing a well-founded model for temporal aspects. Based on the definition from [17], the key components of CPS can be identified as:

- Sensors & Actuators
- Computational System
- Communication

Figure 2.8 shows a schematic overview of a cyber-physical system with the identified key components. These components are divided into the parts "physical world", "cyber world" and "communication" and used to group the appropriate components in the CPS.

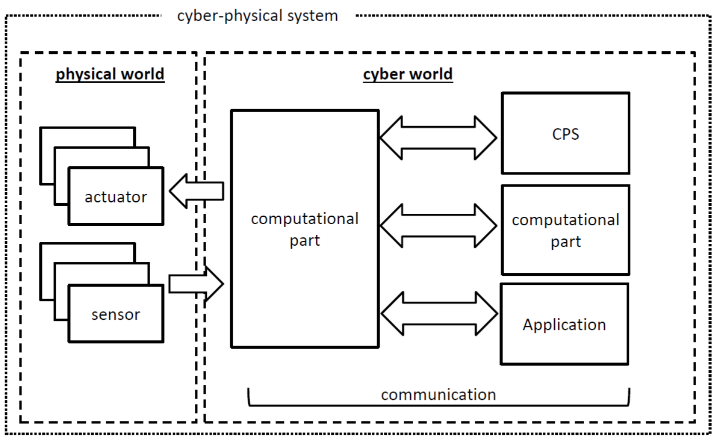


Figure 2.8: Schematic cyber-physical system

Comparison of CPS and RFID

A generic RFID system is depicted in Figure 2.9, compromised of one or more RFID readers, with antennas and tags, a RFID middleware and a RFID application. These components already make it possible to identify the essential parts of a CPS as defined. The *physical world* of CPS is comprised of sensors and actuators. One can think of RFID systems as sensing the presence of objects, which is done through antennas of the RFID readers. In addition, it is also possible to control physical processes through actuators. This is mostly realised through the use of *General Purpose Input / Output* (GPIO) ports. These ports allow to control additional actuators, like electric motors to open doors and gates. The *communication* part in RFID systems is intrinsic and used between the components involved. This can be divided in the communication of RFID tags with the RFID readers using the air interface and in the communication of the RFID readers or parts of the application using a network connection. For example, the RFID

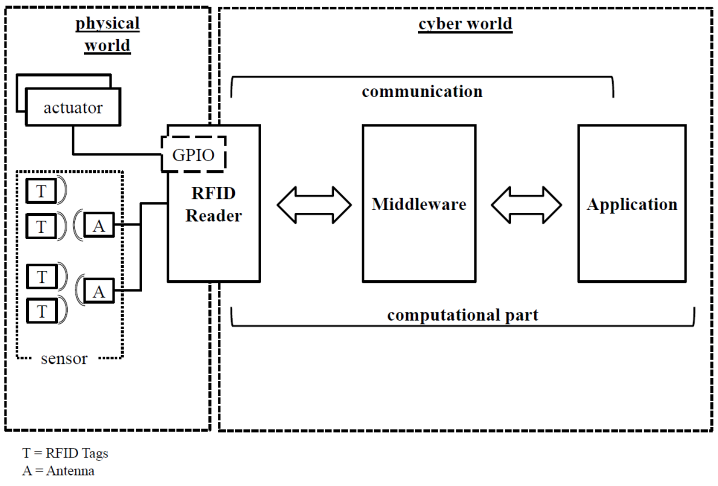


Figure 2.9: Generic RFID System as a CPS

middleware configures the readers and gathers the observed RFID events. These events are then processed in the middleware and filters are applied. Eventually, the processed data is sent to the application and accessible for the users. The *cyber world* is defined by the computational parts. In a RFID system this can either be the reader, the middleware or the application or a combination of those. Depending on the logic of the system, more or less components are involved in the computation.

Nevertheless, the similarities between the central components of CPS and RFID systems already allows to consider most RFID systems as cyber-physical systems. As a result and in regard to testing it might be possible to use this knowledge to adopt some of the techniques used in this area to RFID and vice versa.

3 Challenges of RFID Systems and Testing Requirement Analysis

In order to develop a test methodology for RFID systems it is necessary to gain an understanding of the components of such a system and their functionality. For this reason, the individual components of a RFID system are analysed and corresponding requirements for the test environment and test methodology are determined. In relation to the research questions raised in Section 1.3, the test relevant aspects which must be considered in a test methodology for RFID application testing are therefore identified in this chapter.

This chapter identifies the missing parts in literature, covers the characteristics of RFID applications in the context of testing and serves as a basis for the presented method for test case generation and test automation. First, the architecture of an RFID system is presented and the functionality of the different components are discussed in relation to the test methodology examined in this thesis. This includes the effects of RFID reading events, configuration settings and the interactions of the individual components. Afterwards, the challenges are examined a physical and software level. In particular, defects are classified with a focus on RFID applications, which serves as a basis to determine the requirements for a test methodology. Finally, the various RFID applications are classified in regard to complexity for testing.

3.1 Gaps in Literature

Due to the complexity of software, it is neither technically nor economically possible nor reasonable to test completely considering all possible external circumstances (Patton et al. [32], Myers [10]). This applies in particular to RFID applications. Due to the high complexity of the relationship between components of a RFID system, the application behaviour and its execution context, it is not possible to achieve complete test coverage while taking all possible values of individual test parameters into account. Therefore, functional and non-functional requirements are prioritized in practice and checked against a specification in varying degrees of intensity based on the resulting ranking of individual aspects.

In regard to related work on testing RFID (see Section 2.3.5) an approach which directly addresses functional testing of RFID applications is missing. As identified during the literature review, the aim of other approaches is mainly on performance testing or pure test data generation. Implications of the RFID events on the applications state and, especially, the evaluation of the application state after the execution of tests hast not been part of recent work. Additionally, a sophisticated approach which combines the generation of test cases based the application's logic including the execution and evaluation of the outcome of the test is also missing.

One reason for missing literature on RFID application testing might be the expectation that software testing has been redefined over time into an exact science and is an integral part of software development, but this if far from true. According to [10], software testing seems to be less known than other aspects of software development. Other publications support this statement, for example Tassey [87] has shown that software testing and it's infrastructure is often still inadequate and far from being satisfactory. A recent study of Beller et al. [88] reveals that the majority of developers do not test at all and often there are misperceptions of the amount of time spent for testing.

Especially in the area of RFID, where the main focus still lies on the physical constraints of RFID systems. And truly, it is essential to ensure the correct reading of RFID tags and achieve a high detection rate first. An physical evaluation is the prerequisite for the applicability of the RFID technology and ensures that it can be used in a given business scenario. However, the efforts for this should not be excessive so that the software cannot be tested any more due to budget restrictions and test efforts spend here do not substitute RFID application testing on a system level.

Modelling behavioural aspects of the system is often an integral part of the software development process and therefore does not represent a particular challenge in the application of the concept presented here. The chosen model based test approach is a recognized and often used method for the systematic derivation of test cases. Thus, the basic principle of generating test cases is not unique. Methods for specifying use cases on the basis of activity diagrams and for deriving test cases form them have already been considered in other studies (Section 2.2.2).

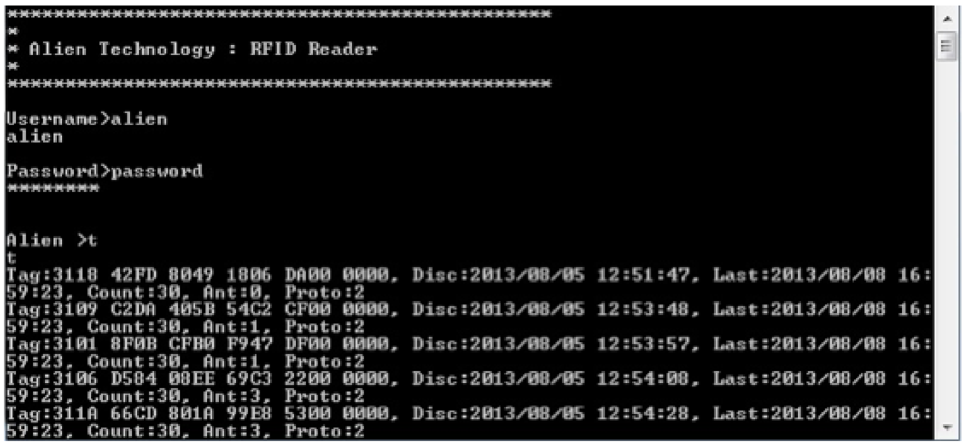
To improve the quality of RFID applications and at the same time to lower the costs of testing a systematic test methodology with high automation potential is needed. However, no known approach covers all the requirements needed for testing RFID applications, such as covering the time lapse during the data acquisition of an RFID system. Since most of the approaches are not combinable or extendable, the approach in this work summarise the most interesting ideas in a flexible, meaningful concept and supplements them with own suggestions for the creation of RFID-specific test cases.

3.2 RFID System Analysis and Architectural Challenges

A typical RFID system consists of a three layered system architecture. Usually a combination of serveral RFID readers (1), a RFID middleware (2) and a RFID application (3). The RFID data, often referred to as RFID event stream, is successively processed on the different layers (1-3). The RFID readers, located at the lowest level, are capable to communicate with RFID tags and therefore build the basis for data capturing. Events collected this way are first processed by the middleware and then passed on to the application. Two fundamental functions are used on each layer to process the data: (1) the actual acquisition and (2) application of filters or aggregations.

When looking at an RFID system, different events are created on the various levels. Therefore, we first divide them into Low Level Events (Tag to Reader), RFID Events (Reader to Middleware) and Application Level Events (Middleware to Application). The RFID events passed from the Reader to the middleware contain important information on which tag was read, when it was read and - depending on the manufacturer's implementation - how strong the signal was. It is possible that additional data may be added to the event during data processing on higher layers of the RFID system. For instance, this could be the name of the RFID reader that has captured the tag (e.g. Application Level Events). Sometimes, this extra information also adds an additional semantic meaning to the event. For example, if a tag is detected at the entrance of the warehouse, this could mean that new goods have been stored.

In order to show a concrete example of an RFID event, an Alien 9800 RFID reader from Alien Technology will be used to take a closer look at the structure of RFID events. This type of reader has been chosen, since the proprietary Alien-Protocol is textual and can be easily interpreted by humans:



```
**********************************************
*
* Alien Technology : RFID Reader
*
**********************************************

Username>alien
alien

Password>password
********


Alien >t
t
Tag:3118 42FD 8049 1806 DA00 0000, Disc:2013/08/05 12:51:47, Last:2013/08/08 16:
59:23, Count:30, Ant:0, Proto:2
Tag:3109 C2DA 405B 54C2 CF00 0000, Disc:2013/08/05 12:53:48, Last:2013/08/08 16:
59:23, Count:30, Ant:1, Proto:2
Tag:3101 8F0B CFB0 F947 DF00 0000, Disc:2013/08/05 12:53:57, Last:2013/08/08 16:
59:23, Count:30, Ant:1, Proto:2
Tag:3106 D584 08EE 69C3 2200 0000, Disc:2013/08/05 12:54:08, Last:2013/08/08 16:
59:23, Count:30, Ant:3, Proto:2
Tag:311A 66CD 801A 99E8 5300 0000, Disc:2013/08/05 12:54:28, Last:2013/08/08 16:
59:23, Count:30, Ant:3, Proto:2
```

Figure 3.1: Example: Alien 9800 Output

Figure 3.1 shows the logon procedure and the execution of the command "t" (abbreviation of "get taglist"). In this case, several RFID tags are located in the vicinity of the antenna. Each tag discovered by the RFID reader is displayed as a separate line containing information about the tag ID, the date on which it was discovered (Disc:) or last detected (Last:), the number of times it was read within the interval, the ID of the antenna on which the tags are present, and the RFID tag protocol (in the example Output "Proto:2" stands for the Air-Interface Protocol EPCC1Gen2).

Even if most RFID systems generate similar data, it depends on the devices used and the querying application (in terms of configuration of data capturing) which data fields are enabled and which functionality is provided. However, a tag event is always a specific RFID tag with a (usually) unique ID (EPC-ID) that was read at a certain point in time on a specific RFID reader (combination of reader and antenna).

In general, the most common RFID event data consists of:

- Source (implicit)
- EPC-ID (or appropriate other identification attribute)
- Time of first sight
- Time of last sight
- Vendor-specific information such as signal strength, count, etc.

Therefore, a RFID reading event can be presented in an abstract form as a tuple:

$$(\text{reader}, \text{id}, t_{in}, t_{out})$$

Typical RFID systems can capture multiple RFID tags simultaneously. For a closer look at the structure of RFID data streams, the example depicted in Figure 3.2 is considered. It shows a

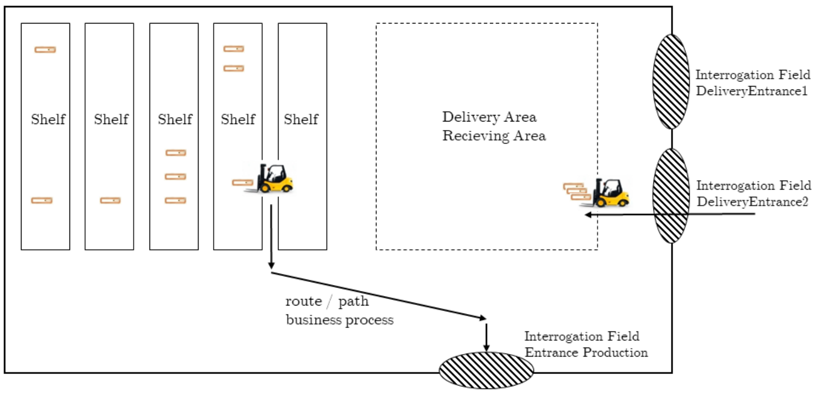


Figure 3.2: Movements of RFID tags in a schematic warehouse

schematic warehouse with two entrances named “DeliveryEntrance1” and “DeliveryEntrance2”. Both entrances and the door to the production area are equipped with fixed RFID readers. The arrows symbolize the forklifts’ movements. At first goods are delivered and recorded by an RFID reader installed at entrance 2. The goods are temporarily stored in the warehouse and after a short period of time, brought into production. Here they are scanned by another RFID reader located at the gate to the production area.

Thus, RFID events are related to place and time. RFID reading events are triggered by objects equipped with RFID tags that move through the room. The location in which RFID events can be generated depends on the reading zone of the RFID reader. In case of permanently installed devices, this location is also fixed. During the acquisition with mobile reading devices which may be attached to forklift trucks, the location is variable. The chronological order of generated RFID events is determined by the route on which the goods equipped with RFID tags move. When this path intersects with a reading zone, the tags are captured. The first and last capture of a tag and the time a RFID tag is located inside the interrogator zone of a certain RFID reader, is determined by the speed of the movement. An RFID event is generated whenever a reading zone is crossed during this movement. An equivalent description of an RFID event is therefore:

$$(id, location, time\_intervall) = (id, reader, (t_{in}, t_{out}))$$

In summary, RFID middleware enables the simple integration of sensors, such as RFID Readers, into enterprise systems and RFID applications and serve both as a filter and aggregation for the increased data volume. In addition, they support extended configuration options to assist application programmers in developing RFID applications. Especially, data management and process management (see 2.3.2) are the essential features that alter the data processed by the RFID application and, hence, is particularly relevant for the test method presented in this thesis.

Challenges for Testing - It can be deduced that the test methodology must be able to describe RFID events and generate them during the test execution. The chronological sequence is determined by movements.

The configuration settings (see Section 2.3.2 and 2.3.2) of the RFID reader and the middleware changes the way in which RFID tags are recognized and thus has a direct effect on the subsequent data processing. Whether the selected settings match the overall system can only be shown by an integration test. A requirement for the test methodology is therefore that the configuration settings can be taken into account.

3.3 Physical and Technical Challenges

Frequent shortcomings of RFID systems are often due to physical problems [109, 110, 111, 112]. Therefore, successful implementation of RFID projects depends greatly on overcoming the physical challenges to achieve reliable capturing of tags. Since, this can often only be coped with by extensive tests and trials before the introduction of RFID, extensive pilot studies are often carried out before the RFID system is introduced in order to investigate its general feasibility.

A huge number of factors, which have a significant influence on the reading range and signal quality, can be identified and therefore are relevant for reading RFID tags. Some of these are listed below:

- frequencies used
- relative position of tags and readers antenna to each other
- polarization of tag and readers antenna

- shape of tag and readers antenna
- radio power
- material the tag is mounted on
- interference (reader-, tag-density)
- coupling (induction, electromagnetic)
- physical disturbances
- tag detection (polling interval)
- velocity of tags
- air interface protocol (aloha protocol)

Some of the factors listed above may change continuously. For example, the reading probability may be close to 100% at first, but it may fall sharply due to changing environmental influences. The insufficient reading rates especially in combination with problematic materials, like metals and water, can be a deciding factor. But also weather conditions can have an impact. An experience from a real-life situation will illustrate this: A feasibility study carried out in autumn revealed a very good reading probability, but it was discovered that several RFID tags could not be read after putting the system on-line. It transpired, that this odd behaviour was caused by pallets which were stored outside the warehouse and therefore were soaked with water. Hence, all sodden pallets could not longer be properly detected.

Currently, there are also limitations with regard to national regulations for international supply-chain flows. For example, there are no standards for worldwide free frequencies that allow cross-border networking of product flows. Also, errors occurring during bulk reading must be eliminated. These reading errors occur particularly frequently when tags and readers from different manufacturers are combined or a huge tag population has to be scanned.

Of course, many of the physical challenges of RFID systems can only be solved on a technical level. A feasibility study must be carried out before the start of a project, as the effects of the radio field are difficult to predict. However, some problems also affect the software level. If physical challenges are not or cannot be tackled properly, for example, due to incorrectly aligned antennas or changed environmental conditions, unintentional reading events can occur. This is probably another reason for the lack of literature on testing RFID applications, since a large part of the testing effort is already invested in mastering these technical challenges. However, for the application, this has serious consequences because either no RFID events are captured or undesired readings are recorded and the corresponding processes (e.g. movement of goods from the warehouse to the salesroom) loose their trustworthiness.

Challenges for Testing - The uncertainty from the unpredictable physics have serious impact on the software. This underlines the importance for software testing and inevitably leads to an increased emphasis on robustness testing. Despite, the fact that unpredictable physical effects also hinder the reproducibility of manual tests, it is necessary to reduce the effort performing these tests. Hence, the approach presented in this thesis, needs to consider an environment that is reliable and controllable in order to perform robustness tests and is capable of repeating tests in the same environmental conditions.

3.4 Software and System Challenges

Although the distributed data processing and the physical challenges of the lower-level components of an RFID system alone justify a test methodology, a closer look at the effects of those on RFID applications is necessary.

Despite the fact that robustness tests play an enormous role in testing RFID applications, the physical challenges also lead to a significant increase in the complexity of RFID applications. Unreliable readings caused by physical effects do not improve in many cases without the use of additional sensors/actuators. However, the use of additional hardware inevitably leads to an even more complex architecture of the overall system.

The fact that the physical challenges cannot always be completely solved is also shown by the way RFID data is described in literature. According to [91] RFID data consists of a data stream with the following attributes:

- redundant data
- grouped data
- moving route
- noisy data

Bai et al. [113] identified three typical scenarios for the reliability of RFID readers: "False-negative" reads are defined as tags that exist but may not be read at all. "False-positive" reading events result in additional and unexpected RFID events. Additionally, "duplicate" reading events are characterized by the fact that they are caused by tags in the reader's field over a longer period of time (i.e. in multiple reading frames), or that several readers are combined to cover a larger area. Keller [114] identified similar effects:

"False-Negativ" - A false negative RFID tag is read if an RFID tag within range of the antenna is not recognized by the reader at all. This corresponds to a pallet that was moved through the RFID portal and unnoticeable loaded onto the truck. Many reasons can be identified for this behaviour. For example, the product types on the pallet have a significant influence on the readability of RFID tags, as water and other liquids (e. g. shampoo) absorb radio waves and thus greatly reduce the reading range. Other reasons are defective or difficult to read RFID tags. To overcome this problem, multiple antennae are often installed to increase the chances of reading a tag. However, this can lead to another problem, the mutual elimination of radio waves by interference effects.

"False-Positive" - In contrast, the term False-Positive RFID tag read has two different meanings. On the one hand, the phenomenon is similar to false-negatives, since physical effects influence the readability of RFID tags. Any metallic material in goods or packaging (e.g. metal foils and metallic paints), the forklift truck itself or anything else in the area of the antennae can unintentionally increase the reading range of the antennae considerably. As a result, tags that are clearly out of reach are read unexpectedly by the reader. On the other hand, the term false-positives refers to tags that have been read and are clearly within the reading range, but for whatever reason they should not be read.

The first question is how to deal with false-positive and false-negative events. Often the middleware and its logic can be used to detect RFID events that are captured multiple times. Particular importance is given to the temporal or location-dependent flow of tags and the associated RFID reading events. This makes it possible to detect erroneous read operations when a business process follows a certain sequence that permits the identification of a order in which tag events occur. For example, lets assume a businesses process which ensures that a tag must be read from reader R1 at first and then it is allowed to be processed by reader R2. If the system recognizes a read on R2 but did not on R1 previously, it can be concluded that something went wrong.

Another way to detect errors without the use of additional sensors would be to track results from business steps [115] at application level. In the business steps “Assembling”, “Packing”, “Repacking”, etc., several tags are put together into larger groups and from then on move together. An example of this is the packaging process, in which several products are palletized. If an RFID tag of a group is now detected, but not all tags belonging to this group, it can be assumed that the reading process was incorrect.

A large number of researchers present different techniques and algorithms for data processing and event stream cleaning to solve the above mentioned problems of capturing tags correctly [113, 116, 117, 118, 119].

Challenges for Testing - Whether the proposed techniques can be applied correctly, becomes evident only when the individual components are integrated. However, this can only be provided by quality assurance methods. Hence, functional testing is an important aspect to ensure quality of the RFID system and needs to be considered for the development of a test methodology.

In addition to data capture, the software architecture of the system plays an important role too. This is mainly due to the distributed nature of the software components involved in processing the data stream, such as the reader itself and the middleware. The software, or rather parts of the application’s logic, of a typical RFID system is distributed over all these components and different configurations of the filters and aggregation mechanisms might be used. In some scenarios, in which it is necessary to share the collected data across different companies, even more additional software such as EPC-IS is needed as well. The resulting software is composed of multiple components and different configuration options, which need to integrate well to perform in accordance with the specification. Therefore, it is necessary to carry out an integration test at system level, since it is the first time all components are assembled. The methodology to be developed needs to be able to validate the requirements for the entire RFID system, RFID middleware and RFID application at the system level.

Since RFID systems aggregate data over time, the test can only be carried out in real time, so it makes sense to automate the tests for reasons of time and cost alone. However, automation can only be accomplished if the RFID environment and its changes can be described properly. Therefore, we derive two requirements for the test methodology: The test methodology must (1) provide a description of a suitable system environment and (2) be able to simulate it reproducibly.

3.5 Classification of RFID Standards

In order to classify the standards (see Section 2.3.3) a survey on the adoption rate of currently available RFID software products offered on the market was conducted. Table 3.1 shows the result of this investigation. Software solutions where no information of supported standards was publicly available are included nonetheless.

Middleware	**LLRP**	**RP**	**ALE**	**EPC-IS**	**TDS**	**RM**	**DCI**
Oracle RFID and Sensor-Based Services	X		X	X	X		
Microsoft - BizTalk RFID	X		X	X	X		
IBM - Websphere Sensor Events	X		X	X	X		
Globranger iMotion Edge Platform							
Odin Easy TAP IBM - Websphere Sensor Events	X		X	X	X		
SAP Auto-ID Infrastructure				X	X		
TrueVUE RFID Platform							
S3Edge Spotlight RFID Server	X		X	X	X		
OATSystems							
RFID Innovations							
RF-IT solutions YOU-R Open Middleware							
Seeburger AG - RFID Slap & Ship Lösung							
CUHK EPCGlobal Middleware			X	X	X		
logicAlloy ALE Server RFID Compliance	X		X	X	X		
Fosstrak ALE Server RFID Compliance	X		X	X	X		
Trancends RIFIDI Edge Server	X		X		X		
Aspire RFID Middleware	X	X	X	X	X		

Table 3.1: Adoption of standards in commercially available RFID software solutions

Summarized, in many of the examined RFID middleware solutions an implementation of the ALE interface can be found. LLRP as a basis for communication with RFID readers is also widely adopted. But some hardware vendors of RFID devices are still using proprietary proctocols, however it is anticipated that the adoption of this standard is progressing in the near future. E.g. Alien Technology, which insisted on using their proprietary protocol for a long time, implemented LLRP for their palette of readers.

With respect to testing, the GEN2 and TDS standards are especially important because they describe the identifiers, which represent the data gathered from the RFID readers. For theoretical considerations of a test methodology identifiers can be substituted by abstract variables, like id_1, possessing the same semantic. However, for the generation of input data and validating concrete RFID events they need to be included. In this approach a black-box test technique has been chosen. Therefore the only way to interact with the SUT is through these interfaces. Hence, input data need to be generated following the definitions of LLRP or other proprietary protocols.

Challenges for Testing - The most important standards in regard to a test methodology are TDS and LLRP, because these standards are used as a interface between the different components of a RFID system and the test environment. As a requirement for a test methodology these need to be considered for the test execution environment in order to allow a practical application.

3.6 Classification of RFID Applications

Depending on the application area of the RFID system, more or less components are involved in the processing of the collected information. There are three types of RFID applications that can be distinguished:

- **Historically oriented tracking applications -** This type usually only stores the data collected by the RFID system in databases. RFID events have no direct influence on the systems behaviour. Typical applications are the tracking of products and deliveries. Unlike the other types, multiple captures of a RFID tag has no direct impact on the application as the observations are just used by means of a journal of movements.
- **Adaptive real-time or control flow & status-based applications -** This class of RFID applications, the data collected by the RFID system changes the state of the system. In other words, it reacts to the tag movements observed. Typical fields of application are, for example, production control or warehouse management. To identify this type of application, it is possible to look at whether it makes a difference if the same RFID tag is captured more than once. For example, an item that has already been seen when entering the warehouse cannot enter the warehouse a second time. Or a product which has already been handled by a certain production step does not need to be processed again.
- **Interaction-based applications -** In addition to the previous type, not only the state of the software is influenced by reading events, but there is also a direct interaction with the environment. An example application can be found in library equipped with RFID. A book that will be borrowed at a self-checkout terminal with RFID requires direct interaction with the user. The user must first start the borrowing process by pressing a button, following several interaction steps between software and user. In regard to the recent trends in the field of “Industry 4.0”, RFID applications of this category are expected to increase.

In comparison to other approaches which mainly focus on test data generation [26, 30, 31, 91, 102, 101, 27] interaction-based applications in particular have not been addressed properly.

However, especially this type of RFID application increases the complexity of the tests through the various interactions. In this thesis a model based test approach is introduced which is capable to capture these interactions and at the same time can be refined with additional test data (e.g. test oracles) to test and evaluate such applications.

3.7 Conclusion

Due to the complex and distributed architecture of RFID systems, system level testing can be seen as an important aspect to improve the quality of the RFID application. Especially as there are many factors, like configuration settings of the individual readers and the middleware, which influence the correct interaction of the sub-systems and can only be tested when all components are finally composed to a complete solution.

RFID systems are unreliable in certain conditions and composed by many components. Timing and the order of RFID events are essential for distinguishing logical RFID events. The use of RFID can offer valuable benefits, but also comes along with many challenges the application needs to be prepared for. Considering the identified issues of the previous chapter in the context of this thesis, typical effects of RFID on the overall system are:

- False or unintended events (false negatives, false positives)
- Unrecognised events that reappear later need to be handled accordingly (robustness design, error handling)
- Interaction between the various parts of the RFID System (system testing)
- Suitability of the chosen configuration options (filter, aggregations) in regard to the business processes (timing, location)

In this chapter the challenges of RFID applications have been explored and requirements in regard to a test methodology have been derived based on these observations. Furthermore, different types of RFID applications have been classified which can be tested with the methodology for testing RFID applications presented in this thesis. The central question remains whether the software solutions can cope with the numerous challenges at the physical and software level and at the same time also implements the specified functionality properly. On the knowledge gained during the analysis of the RFID systems and in regard to the research question how a RFID application can be tested, a test methodology was developed that is able to meet the identified challenges.

To achieve this, a semantic model to describe the system environment and the hardware was created. With this abstract model its possible to describe the components of an RFID system and the according movements of RFID tags, respectively input data for the SUT. A systematic method for test case generation including constraint based test data selection and in order to evaluate the test results a semantic representation of the application back-end is presented.

Additionally, a automatic test execution environment, capable to simulate the unreliable capture of RFID tags and control the environmental conditions of an RFID system is proposed.

4 ITERA Test Methodology and Model Based Testing of RFID Applications

Chapter 3 examined the typical architecture of RFID systems and showed that particularly the physical challenges have a significant impact on the development, operation and testing of RFID applications. These effects also manifest themselves in an increased complexity for all quality assurance measures. So far, however, there are hardly any methods and tools that meet the special requirements of testing RFID applications (Section 2.3.5, 3). Therefore, a constructive model-based concept for determining tests, as well as a tool-kit (Section 2.3.4) for automated test execution including an RFID environment simulation will be presented.

This chapter covers the definition of the semantic model, the ITERA meta-model and the concrete application of the ITERA method for creating RFID application tests from the specification.

The formal definition, notation and the semantics of a RFID system as well as a model of the application are described in the first part of this chapter. An abstract model based on set theory is introduced that describes the RFID readers, RFID tags and the processes of an RFID system.

The second part of this chapter contains a test methodology to systematically obtain test cases. For this purpose use cases of a RFID application are identified and analysed. Each use case is successively refined by activity diagrams, which are used as a basis for augmenting contextual test data and systematic test case derivation. The combination of both, the semantic notation and the activity diagrams of the use cases allow the generation of executable tests for RFID applications suitable for automated testing.

4.1 ITERA Methodology

The introduction of RFID can provide a lot of benefits to businesses, but also comes along with many challenges. Based on the findings of the RFID system's analysis in Chapter 3 and in order to answer the research questions raised in Chapter 1, this chapter presents a novel test methodology, that not only meets the challenges identified, but also allows for the automatic and systematic derivation of test cases.

The ITERA method is based on the premise that initially rough business process models can be created from use cases and successively refined by enriching them with increasingly detailed information from the requirements specification and the environment (see Section 4.4). Through enrichment with test relevant information it allows for the generation of test cases. In the narrower sense, test relevant information may be user input, system assertions and context parameters. Technically, the refinement is realised through the integration of test-specific details using a specially tailored UML profile, called the ITERA meta-model (Section 4.3). The test model is then used to systematically extract test paths (Section 4.5). Those paths in combination with test data constitute the tests (Section 4.6). These tests are then supplemented by an automation tool (see Chapter 5), which can execute the tests in a simulated environment without the need of manual intervention. This simulation covers the manipulation of the SUT and its environment, in particular by temporal aspects and the stimulation of the SUT by artificial RFID events.

Similar methods for test generation have already been investigated in the scientific literature. While the individual approaches differ in detail, there is agreement on their basic feasibility. Despite the differences in the individual steps of test case generation, those approaches largely follow a uniform pattern (Section 2.2.2), on which the methodical approach presented here is also oriented.

4.1.1 Overview of the ITERA Methodology

Integrative Test-Methodology for RFID Applications (ITERA) is a test method, a semantic model of the RFID environment and a test environment for RFID applications. The method presented is used to gradually transform use cases developed in cooperation with stakeholders into a semi-formal test specification. Test cases are then systematically determined from the models created, in order to execute them in the test environment eventually. ITERA, thus, not only enables the specification of RFID tests, but also allows for describing the RFID environment with the aim of performing automated software tests.

The methodology presented in this thesis is based on three models: The domain model (class diagram), the test model (activity diagram & meta model) and the physical model. Each model represents one aspect of the SUT. The most important model in the ITERA context is the process model (i.e. test model) based on UML activity diagrams. It covers the dynamic processes of the RFID application and allows for the systematic generation of test cases. The domain model is a static model that is used to identify domain terms and the dependencies between objects and helps to identify test data. A UML class diagram is used for this purpose. The environmental model helps to capture the spatial and temporal aspects of the underlying physical processes. It is mainly used as a tool to enhance the UML activity diagrams, since these do not allow for a sufficient description of temporal aspects, as needed for testing RFID applications.

Figure 4.1 provides an schematic overview of the ITERA methodology for RFID application testing. First, several models representing the different use cases are created and successively refined using the specification of the SUT. In a second step, test cases are derived which can also

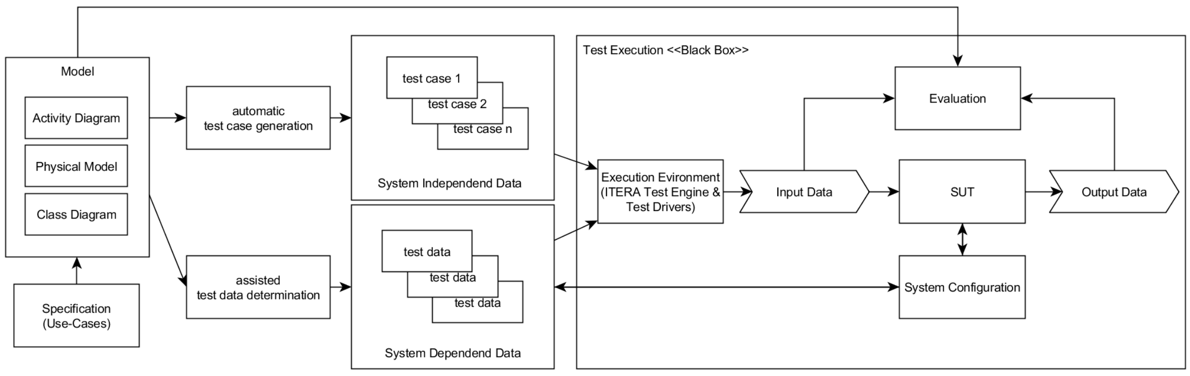


Figure 4.1: Schematic overview of the ITERA Methodology

be used to determine the test data needed for test execution. Once these steps are performed, the tests can be executed using an RFID environmental simulation in order to evaluate the SUT.

4.1.2 ITERA Phases

The concept elaborated in this thesis consists of four phases and follows the procedure of Model Based Testing (MBT) (Section 2.2.2). These comprise of:

1. **Modelling of behavioural aspects**
   In the first step, behavioural aspects of the SUT are modelled. The presented approach supposes that requirements are available and that these can be transferred into a behavioural model in form of an UML activity diagram. This restriction is based on the concrete implementation of the method in this thesis, but in principle it can also be applied to other models.
2. **Refinement of the models with test-specific details**
   In a second step, the behavioural model, i.e. the UML activity diagram, is enriched with test-specific details. These include test-relevant business rules and in particular an RFID context model. At the technical level, this enrichment is realised by an UML profile, which extends the UML meta-model with the ITERA context model. This model can be extended and adapted in order to meet special features of components involved in an RFID system that are not considered in this work.
3. **Transformation of the test model into tests**
   By applying a model transformation, the UML activity diagram enriched by the ITERA context is transferred into an abstract test case model, which extracts test-relevant aspects of the behaviour model in the third step. Thus, it forms the basis for the determination of concrete test data and for further processing towards executable tests.
4. **Automated execution of tests by simulating the RFID environment**
   Through combination of test cases and test data, concrete test cases are created in the fourth step. It is not a prerequisite that the abstract test cases are generated from a UML

activity diagram. Test cases created manually using elements from the ITERA meta-model are also valid input that allow for creating executable tests.

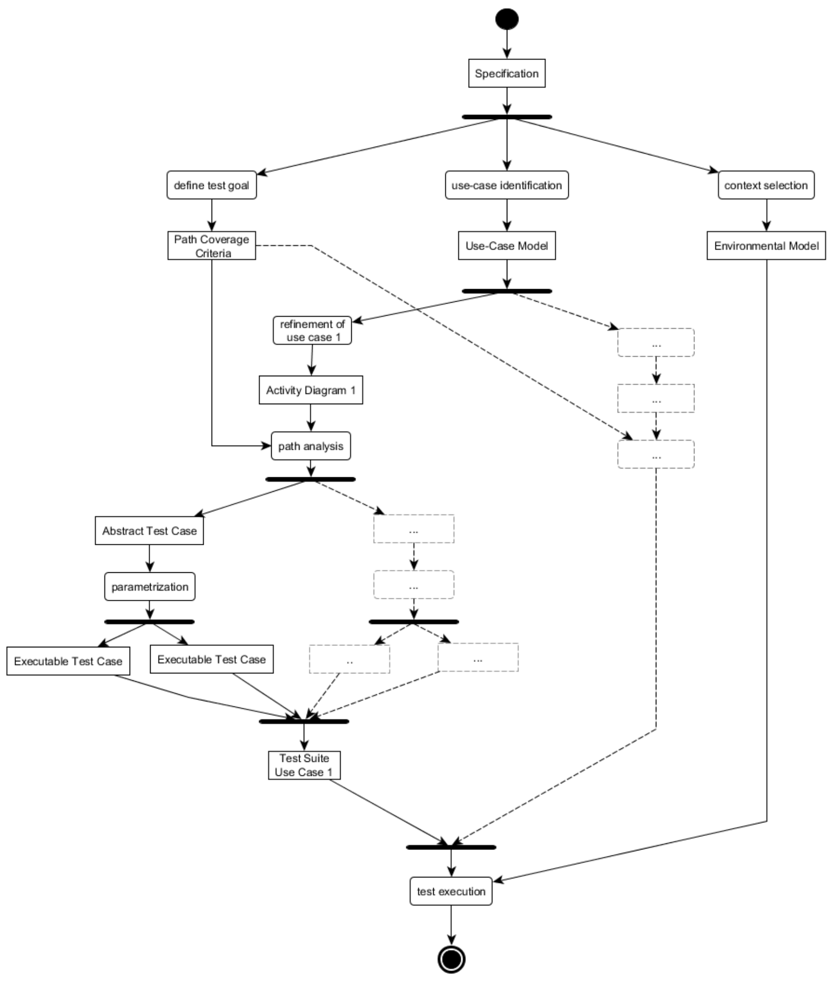


Figure 4.2: ITERA test case generation process flow

Figure 4.2 exemplary visualises the flow of activities to generate a set of tests within the ITERA methodology. It starts with the specification, which is used to define a test goal, identify the different use cases and to determine physical parameters. The definition of a test goal includes a coverage criteria, which will be used as a parameter for the automatic extraction of test cases through the path analysis performed for each use case. The context selection captures additional testing parameters, like timing and run-time specific details of RFID reading devices (e.g. type of reader, IP address, etc.) needed to set-up the test execution framework. The use cases that are identified from the specification are used as a basis to model activity diagrams (phase one of the ITERA methodology). Each activity diagram will then be refined by using the elements from the meta model as defined in the second phase. The path coverage criteria is used in the third phase to extract the abstract test cases. Enriched with additional run-time specific test data each abstract test is converted to an executable test case, which can be used within the test framework. As each use case might contain several execution paths (test cases), these are combined to a test suite. With additional parameters covered in the environmental model and test data selected by the tester the execution of the tests can be carried out (phase four). The process to generate test cases in Figure 4.2 is not necessarily linear and should be applied iteratively to improve the generation of tests.

4.2 Semantic Foundation for Modelling RFID Systems

In order to be able to specify test cases for testing RFID applications later on, a convenient semantic of the application and the RFID system's components is required.

The first step towards testing of RFID-based applications is to define this semantics for RFID applications, RFID readers, RFID tags and their interactions. This semantics for elementary RFID interactions is based on set theory and a discrete time model. The following chapter introduces an abstract but formal semantics, in order to describe the different parts of a RFID system without the need to specifically address the different RFID readers, RFID middleware or tags used in the actual implementation and realisation. The preliminary work of this subsection has also been presented in [120, 15].

4.2.1 Discrete Timing Model

To simplify the notation, a model of discrete time is used. So, the sequence $T = t_0, t_1, t_2, \ldots$ describes different time steps, where RFID related actions might happen. An identifier of a RFID transponder can be described as id and a RFID reader as R. Thus, a certain RFID tag can also be written as id_2 and a distinct reader can be written as R_1, where the indices are used to distinguish multiple readers or tags. This is a sufficient abstraction level for the specifications used in the test methodology.

Regardless of the continuous progression of time in the underlying physical processes, the RFID readers are just capable of sensing RFID tags in their field at these discrete points. This assumption is underlined by the fact, that there are usually no incomplete readings of a tag's identifier. Incomplete transmissions are prevented by the air protocols of the RFID readers, which are using error checking mechanisms [81]. In contrast, considering a continuous model of time, there could be state changes observed in-between the discrete points of time. However, this is not the case from a RFID reader's point of view. Figure 4.3 shows the progression of four RFID tags id_1, id_2, id_3, id_4 through a reader's radio field over time. The associated changes of the reader R_1 using a polling command to retrieve the captured RFID tags are also depicted.

Regarding the example in Figure 4.3, state changes can be observed at t_{0-4} and t_{6-7} only. Time points without changes, like t_5 or t_8, can be skipped entirely, since there is no information gain compared to the time step before (t_{i-1}). Therefore, the use of a discrete time model is reasonable and sufficient for this approach.

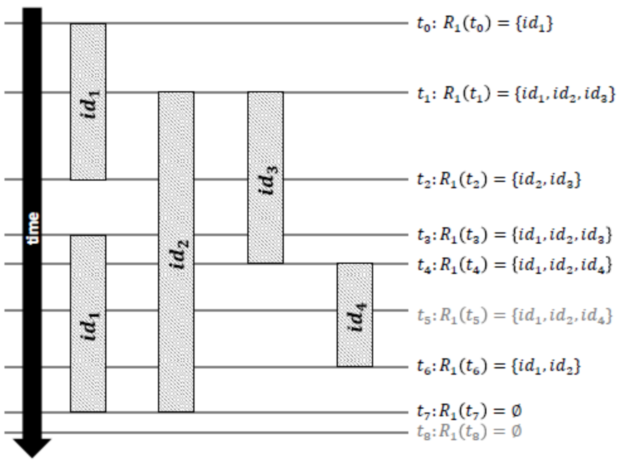

time
id_1
id_2
id_3
id_1
id_4
t_0: $R_1(t_0) = \{id_1\}$
t_1: $R_1(t_1) = \{id_1, id_2, id_3\}$
t_2: $R_1(t_2) = \{id_2, id_3\}$
t_3: $R_1(t_3) = \{id_1, id_2, id_3\}$
t_4: $R_1(t_4) = \{id_1, id_2, id_4\}$
t_5: $R_1(t_5) = \{id_1, id_2, id_4\}$
t_6: $R_1(t_6) = \{id_1, id_2\}$
t_7: $R_1(t_7) = \emptyset$
t_8: $R_1(t_8) = \emptyset$

Figure 4.3: Example progression of tags through the reading field of an RFID Reader

Based on the information presented, only changes of the sets are essential. However, the exact points of time, where RFID events might happen, still need to be identified. This can be done by an analysis of the RFID events shown in the example. The result of this analysis shows, that each time point where a tag enters, respectively leaves, the interrogation zone of a RFID reader needs to be identified, to build the discrete model of time. Nevertheless, the time passing by in-between two time points can be variable.

4.2.2 RFID Readers and Tags

The focus of this thesis is to test functional aspects of the software. Generally, it is not possible to capture the software in every detail, so a more abstract model that still captures the essential parts is needed. For this purpose, a semantics and notation based on sets is introduced, to be able to specify the tests.

A generic RFID system is defined as:

Definition 4.2.1 (RFID system)
A *RFIDSystem* is a tuple of sets containing the physical RFID readers and RFID tags.

$$\mathit{RFIDSystem} = (\mathit{Readers}_{sys}, \mathit{Tags}_{sys})$$

All distinguishable tags of a RFID System are collected in a set of RFID tags and defined as:

Definition 4.2.2 (RFID tags)
A set of tags $\mathit{Tags}_{sys} = \{id_1, id_2, \ldots, id_n\}$ is a set, of distinct RFID tags with unique identifiers. $n \in \mathbb{N}$ is finite and represents the count of RFID tags in this system.

Following the common definition of a well defined set, a set is a collection of distinct objects. So, if $id_2 \neq id_5$, the set $\{id_2, id_5\}$ describes two distinguishable RFID tags, which can be separately read and are placed on two different objects.

A typical RFID system consists of at least one RFID reader, but more commonly more than one reader can be found in industrial set-ups. Therefore, the set $Readers_{sys}$ is the collection of all RFID readers the RFID system consists of.

Definition 4.2.3 (RFID Readers)
The set of RFID readers $Readers_{sys} = \{r_1, r_2, \ldots, r_n\}$ is a collection of distinct RFID readers the system consists of. $n \in \mathbb{N}$ is finite and represents the count of readers in this system.

E.g. $Readers_{sys} = \{r_1, r_2\}$ is a RFID system consisting of two different RFID reading devices, in this case called r_1 and r_2. Where the index of r_1, r_2 denotes the identifier of said RFID reader.

The state of a RFID reader can also be described by a set of RFID tags, which are in the detection range of the reader.

Definition 4.2.4 (RFID Reader)
Every RFID reader can be described as a function *reader* mapping the RFID tags a reader can sense while inside of the reader's radio field.

$$reader\colon Readers_{sys} \to \mathcal{P}(Tags_{sys})$$

$$reader : r_n \mapsto ID$$

- Where $r_n \in Readers_{sys}$ specifies the RFID reader;
- and $ID \subseteq Tags_{sys}$ is a set of RFID tags which are in the readers sensing range.

E.g. $reader(r_1) = \{id_1, id_5\}$ denotes reader 1 with two tags in its sensing range. To simplify this notation, the index n of the reader r_n can also be annotated as an index to the function name, so

$$reader_1 = \{id_2, id_5\}$$

describes the set of RFID tags which are detected by $reader_1$. In this example, the set contains the tags id_2 and id_5.

However, the tags that are present at the time of a so-called inventory operation (reading tags in the radio field of a reader) can change over time. Hence, the function of a RFID reader is refined by adding an argument t expressing a distinct point in time to reflect the tags present at the specified point in time.

Definition 4.2.5 (Refinement of RFID reader)
The RFID tags present in the reader's radio field at a certain point in time can be described by a function mapping the reader and the discrete point of time to a set of tags.

$$\mathit{reader} : \mathit{Readers}_{sys} \times T \to \mathcal{P}(\mathit{Tags}_{sys})$$

$$\mathit{reader} : (r_n, t) \mapsto \mathit{ID}$$

- Where $r_n \in \mathit{Readers}_{sys}$ specifies the reader;
- $t \in \mathbb{N}$ denoting the discrete point of time;
- and $\mathit{ID} \subseteq \mathit{Tags}_{sys}$ is a set of RFID tags.

As before, for simplicity we can use the index n of r_n as the index of the reader function. E.g. $\mathit{reader}(r_1, t_3)$ is equivalent to $\mathit{reader}_1(t_3)$. According to this simplification it is possible to write:

$$\mathit{reader}_1(t_3) = \{id_2, id_5\}$$

Here reader_1 has two tags present in its detection range at t_3, namely id_2 and id_5. If there is no RFID tag in range or RFID tags are not detected during a inventory operation, the function returns an empty set ($\emptyset$). E.g. if there are no tags present at t_0, the result of the function of reader_1 would look like $\mathit{reader}_1(t_0) = \emptyset$.

Since most RFID applications are business based, an abstraction function might be required, where comparable products have to be identified or certain attributes of an *Object* might be needed.

Definition 4.2.6 (Abstraction Function)
A function *abstraction* allows to identify or obtain certain attributes of a object, and is defined as:

$$\mathit{abstraction} : (\mathit{attribute}, \mathit{object}) \to \mathit{value}$$

Usually, this is realised by a combination of the object's attributes. E.g.

$$\mathit{abstraction}(id_2, \mathit{type}) = \mathit{pullover}$$

describes that the tag id_2 is attached to an item of type pullover. Or in a more operational way:

$$\mathit{type}(id_2) = \mathit{pullover}$$

To obtain the RFID identifier of the tag id_2, the abstraction function can be used this way:

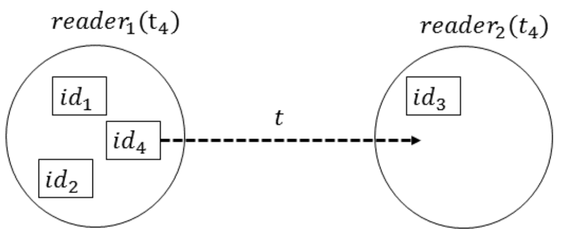
$$\mathit{tagID}(id_2) = 0001$$

The identifier of a RFID Tag usually contains an EPC-ID, but for simplicity a simple numeric description (in the example above 0001) is used to improve readability.

This simplification is suitable for the testing purpose. But, whenever an identical RFID tag identifier is used on two different products, the abstraction function does not allow to distinguish between the products. Hence, problems for the RFID system might occur when trying to recognise the products solely by their identifiers. However, these kind of problems can only be solved by additional sensors or human interaction, which is out of the scope of this work and will not be considered any more in this thesis.

4.2.3 RFID Events

As shown in the previous section, every RFID reader can be described as a set of RFID tags, which are currently in its reading rage. In typical RFID systems, however, these tags are not stationary and move from one reader to another. In order to describe so called low-level RFID events, a consequence of temporal and spatial changes of physical objects, the changes of sets over time need to be considered in this model as well.



t_n	$reader_1$	$reader_2$
t_4	$\{id_1, id_2, id_4\}$	$\{id_3\}$
t_5	$\{id_1, id_2\}$	$\{id_3\}$
t_6	$\{id_1, id_2\}$	$\{id_3, id_4\}$

Figure 4.4: Schematic illustration of tag movement

Figure 4.4 shows two RFID readers, $reader_1$ and $reader_2$, and the RFID tags in the reading range of the antennas at t_4. As already presented, the tags, which are in the detection range of $reader_1$, respectively $reader_2$, at t_4 can be described as defined: $reader_1(t_4) = \{id_1, id_2, id_4\}$ and $reader_2(t_4) = \{id_3\}$.

Since RFID readers are only capable to sense RFID tags in their reading range, only the occurrence of tags in a reader's field of sight can be used to model the input of RFID systems. E.g. at the transition between t_4 to t_6 the tag id_4 is moved to $reader_2$. Once it leaves the sensing range at t_5, it disappears and is in neither of the two sets. Eventually, id_4 reaches $reader_2$ in t_6.

Thus, a tag movement also needs to be modeled by adding or removing elements to the sets. The arrival of a tag can be expressed by:

$$id_4 \notin reader_2(t_i) \wedge id_4 \in reader_2(t_{i+1})$$

Or in a more operational way:

$$reader_2(t_{i+1}) := reader_2(t_i) \cup \{id_4\}$$

Whenever the tag id_4 has been in the reading range of $reader_1$ at t_i and will not be read at t_{i+1}, this departure of a tag can be described by:

$$reader_1(t_{i+1}) := reader_1(t_i) \setminus \{id_4\}$$

The low-level *RFIDEvent* is a combination of both transitions, meaning adding and removing a RFID tag. In reality, however, time passes while a RFID tag moves through the reading field. Therefore, a RFID event contains the arrival and the depature of the tag, in relation to the reading zone.

Definition 4.2.7 (RFID Event)
A low-level $RFIDEvent := (reader, tagid, t_a, t_d)$ can be described as a tuple, where

- *reader* denotes the reader,
- *tagid* the identifier of the RFID Tag,
- t_a the arrival time
- t_d the depature time

To describe a RFID event, it is not always necessary to specify both time points explicitly. It is sufficient if the period of time in which the RFID tag is within the reading field is given, as this time frame corresponds to $duration = t_d - t_a$.

$$RFIDEvent := (reader, tagID, duration)$$

With this description, it is now possible to describe the dynamic nature of RFID data more precisely by reading events similar to typical RFID readers. The set-based semantics introduced here already allow a abstract description of elementary parts of the RFID system that are necessary for tests. It can be used to describe the RFID systems and its components, consisting of RFID readers and RFID tags, and whether a tag is within the sensing range of a reader at a certain point in time. In terms of testing, the reading events play are an important role, because they represent the input data of the application to be tested. The software components have not been considered in the first part of this chapter, but will be considered in the next section.

4.2.4 RFID Applications

The application states can be modelled similar to the RFID system by using sets and relational algebra. In the context of this work the states of an application are considered as something that is altered by the processing of RFID events. The basic principle behind the states is to identify meaningful sets on which the decisions of the business operations can be based on. However, as these states are usually very application specific there exists no general method of obtaining them. Nevertheless, an indicator to identify the different states of an application can be to consider multiple reading processes and their results.

For example, all goods that are currently located in the warehouse or sales-room can be used to form a set of elements containing all goods (tags) in the warehouse, respectively in the sales-room. If an object with an attached RFID tag is moved into the sales-room, a RFID event is processed, which changes the elements of these sets. Another example would be all books in a library that are borrowed or have already been borrowed. A book that was previously borrowed cannot be borrowed a second time and would result in an error during RFID event processing.

The relational algebra [121, 122] consisting of e.g. selection, projection, Cartesian product, set union and set difference is a suitable instrument to describe them. One way to create such sets is by analysing the database of the application. A simplified example of an application's database is given in Table 4.1, taken from a simple retail store.

TagID	ProductType	State
0001	sweater	warehouse
0002	sweater	warehouse
0003	sweater	salesroom
0004	t-shirt	salesroom
0005	t-shirt	sold

Table 4.1: Table “stock” of a demo application's database

An analysis of the table allows to identify representing states (warehouse, salesroom, sold). It can be used to build the appropriate sets to describe the application's state. This can be realised for example by using a *selection*, which allows for selecting entries in the database and transform them to a set. Here the previously defined *abstraction* function is used to build a set of all entries in the database fulfilling a certain attribute.

$$state : TagID \rightarrow State$$

E.g. the sweater with the TagID 0001 has the attribute *warehouse* according to Table 4.1:

$$state(0001) = warehouse$$

This allows grouping objects (products) with the state warehouse in a set W, which contains all products in the *warehouse*. According to this decision, the example database can be divided into three different sets, namely *warehouse*, *salesroom* and *sold*:

- Items in warehouse:

$$W := \{TagID \mid state(TagID) = warehouse\}$$

- Items in salesroom:

$$S := \{\mathit{TagID} \mid \mathit{state}(\mathit{TagID}) = \mathit{salesroom}\}$$

- Items in sold:

$$O := \{\mathit{TagID} \mid \mathit{state}(\mathit{TagID}) = \mathit{sold}\}$$

Hence, Table 4.1 can be transformed to the following concrete sets at t_0:

$$W(t_0) = \{0001, 0002\},\ S(t_0) = \{0003, 0004\},\ O(t_0) = \{0005\}.$$

Sometimes, however, it is not trivial to identify different states because they can span several tables or have difficult relationships among each other.

4.2.5 RFID Business Processes

When it comes to RFID applications and their use in industry, business processes are used to describe the tempo-spatial nature and the dependencies between the movements of objects, which can be observed by RFID readers, and the expected changes of the application. Business processes where RFID events change the state of the application are of particular interest. It is essential to integrate these movements and their meaning into the test models, since they allow the mapping of observed reading events to status changes in the software. This mapping is an important aspect of every software test, especially when considering black-box testing and can be realised by a combination of the reader input model and the application model introduced earlier.

The changes of the sets, which are a result of RFID events, can be further modelled as operations on these time based sets, similar to the approach for the RFID reader input. E.g. a product with the associated tag id_1, which was not in the warehouse at t_i, but will be added to the warehouse in t_{i+1}, can be described by

$$id_1 \notin W(t_i) \wedge id_1 \in W(t_{i+1})$$

Or in a more operational way:

$$W(t_{i+1}) := W(t_i) \cup \{id_1\}$$

To be able to describe all state changes of the application, this approach distinguishes three operations for a generic product p:

- add $\equiv W(t_{i+1}) := W(t_i) \cup \{p\}$
- remove $\equiv W(t_{i+1}) := W(t_i) \setminus \{p\}$
- contains $\equiv p \in W(t_i) = \mathit{true}$

An example business process *delivery* could be: Once goods are delivered to the warehouse and pass a RFID gate ($reader_1$), the detected products need to be added to the warehouse.

If a tag id_2 is sensed by $reader1$ at t_4, the discovered product, associated with the tag id_2, needs to be added to the set W. This can be described according to the introduced notation as:

$$reader_1(t_4) = \{id_2\} \implies W(t_4) := W(t_3) \cup \{id_2\}$$

In this section a novel set-based semantics with a discrete time model was presented. The model also enables the description of the RFID system, consisting of RFID readers, RFID tags and RFID application, as well as its current state. These semantics build the foundation for the ITERA meta model introduced in the next section.

In principle, inputs of an RFID application and their state changes can already be modelled using this notation. The status changes of the application are triggered by RFID events. These events originate from business processes that involve physical movements of RFID tags. Essentially it is already a suitable basis for describing black-box tests.

In the following sections, a closer look at the individual steps of the ITERA test method will be given. At first, a meta model is presented to further refine the created activity diagram and to augment it with test-relevant information. Subsequently, an activity diagram is created based on a example use case. This serves as a starting point for determining test cases in the following sections. A model transformation of the activity diagram leads to an *ActivityGraph*. Through traversing this graph, execution paths of the modelled application case are derived. Each of these paths is then transferred to a test and manually enriched with test data. Finally, robustness tests are created based on the existing test cases.

4.3 ITERA Meta Model

The goal of this thesis is to generate test cases for RFID applications from previously created activity diagrams of the use cases and then execute them automatically and in a defined operational environment. This assumes that the model used for test generation contains the necessary context parameters in addition to the domain-oriented test data. However, UML does not provide corresponding modelling elements. Therefore, it is necessary to define or extend the modelling language with an appropriate meta model that provides these modelling elements. The following sections explain the meta model for test modelling and the necessary steps during the refinement of the system behaviour, which will enable the application of the test methodology later on.

In this thesis the point of view a tester is considered and the system is treated as a black-box. A test contains precise instructions about required user input and expected results. Based on this, a tester can check the correct behaviour of the SUT with regard to the realisation of the business process. From the tester's point of view, only process steps that require inputs from the user or that deliver output to the user are relevant.

4.3.1 Overview of the ITERA Meta Model

The purpose of the ITERA meta-model is to enhance the activity diagram with test-relevant aspects and to provide a interpretation of the actions modelled within the activity diagram. This enables the automated execution of test cases derived from the test model and makes it easier to determine the domain specific test data for a test case.

ITERA supports the modelling of context parameters and test relevant data, including inputs and expected outputs, within the framework of model based testing. The integration of these parameters into the test model is part of an iterative refinement of the previously created activity diagram. This is achieved by using a UML Profile (extending the OMG Meta Object Facility [47]). For the design of the meta-model, the modelling language UML was chosen, because it ensures that the test models based on this meta-model can be easily integrated into the development process of a software system. In addition to test modelling, the meta-model also bridges the gap between the semi-formal UML activities and the semantic model from Section 4.2.

The ITERA UML profile provides stereotypes that enable a direct modelling of the test relevant aspects into activity diagrams. This eliminates inconsistencies caused by separate documentation of additional test data. It is also suited for automated test execution, as all data is available through the models. The generation process of executable test cases is divided into two stages. At first the additional information contained in the activity diagram is transferred to abstract test cases. In the second step input data is provided which manifests itself in the concrete test cases, which can be used by the test environment.

Currently, UML activity diagrams do not directly provide a suitable way of including additional test data. This is partly due to insufficient semantics of the activities and their informal descriptions (e.g. "place book"). The UML specifies the semantics of an action as:

> "An Action is a fundamental unit of executable functionality contained, directly or indirectly, within a Behavior. The execution of an Action represents some transformation or processing in the modeled system, be it a computer system or otherwise." [47, p. 443]

The semantics of the individual activities in regard to testing is not apparent, especially considering the high abstraction level of the activity diagram created from the use case. Therefore, it is necessary to clarify these with regard to the derivation of test cases. This is in accordance with the UML specification, stating that one reason for extending UML with profiles is: "Add additional semantics to UML or specific metaclasses" [47, p. 252].

By looking at an application from a black-box perspective, the behaviour is determined by inputs and outputs. Their type is defined by the interfaces of the SUT and informally described by the text contained in the activities of an activity diagram. Consequently, an activity diagram created from the application specification already contains descriptions of inputs and outputs, but usually in an informal and not computable format.

Common to all RFID systems is that RFID tags are detected when they pass through the interrogation zone of a RFID reader. However, depending on the implementation, interactions with other interfaces may also be possible. For example, interaction-based RFID applications (see Section 3.6) usually also offer a graphical user interface that can be used for interactions with the system.

As there are numerous different application areas for RFID technology, the number of available interfaces is difficult to delimit. For example, light barriers, weigh scales or other sensors can be employed also. However, a comprehensive description of all interfaces is too extensive and would go far beyond the scope of this work. For this reason a restriction to the aforementioned graphical user interface was made, in addition to the RFID components. In principle, other interfaces can be described similarly. An extension of the presented methodology by any input or output type is allowed, provided that appropriate changes are also made in the subsequent steps of the methodology and the test automation technology supports it.

An activity diagram also contains decisions that manifest themselves in alternative control flows and usually result in different execution paths of the SUT. For the transformation of UML activity diagrams to test cases, the distinction between alternative control flows initiated by the user or by system-internal functions is decisive. System decisions are often significantly influenced by user input and reflect the functional requirements of the application to be tested. However, internal system functions can also completely elude external influence, because they use parameters that cannot be determined by the user. Automated test case generation for such systems is expected to generate valid test cases for all alternatives in the control flow of such decision nodes. From the black-box perspective, the expected behaviour of the SUT can only be expressed through business rules (including preconditions). In order to influence the control flow in one direction or another, well selected test inputs must be chosen which either fulfil the precondition or not. In contrast to the system-internal decision nodes in the control flow graph, the decision criteria of user inputs are not determined by preconditions, but by alternative ways of interacting with the system interfaces. User decisions can be non-deterministic but constitute a fundamental part of the test, as all possible ways the user can interact with the SUT need to be tested. Likewise, a UML profile must be able to model the dependency of certain control flow alternatives on arbitrary decisions of the user.

However, the exclusive consideration of the system's in- and output sequences is not sufficient to determine RFID specific test cases from activity diagrams. In contrast to other works that generate test cases from UML activity diagrams, RFID application tests must also consider activities that only have an indirect correlation to input. These are characterised in particular by the temporal sequence of activities. Depending on the technology used and the configuration of the RFID system, an RFID tag can be read several times and thus results in multiple inputs to the SUT. Therefore, the test cases created from an activity diagram must provide detailed information about the temporal aspects (e.g. sequence, duration) and these need to be reflected by corresponding elements in the activity diagram.

Summarising the deduced requirements for an activity diagram, which can be used as a suitable test model:

- **Requirement 1:** The description of business processes modelled with UML activity diagrams usually include user-specific and system-specific process steps. User-specific process steps are all types of decisions or actions and interactions with the system the user can carry out. From a classical black-box viewpoint, the user can only submit input to the system or observe the corresponding outputs of the system. Therefore, the meta-model as basis for the test model must be able to distinguish between actions initiated by users and actions initiated by the system.
- **Requirement 2:** Activities within the UML activity diagrams usually have no explicit semantic. In order to use it as a solid basis for test case determination, the meta-model for testable activity diagrams needs to be extended with a profile that also provides modelling elements for the semantics of these activities. These are *interactions* and *decisions* for each user and system.
- **Requirement 3:** Usually the duration of a sequence of activities is not considered when using activity diagrams. Therefore, the meta-model must be able to cover timing aspects as well, especially since it is an important aspect of RFID system testing.

4.3.2 Class Diagram of the ITERA Meta Model

Inputs are provided through user interactions with the system. In contrast, outputs are system interactions with the user. In the following sections, a separation of actor and system is made. Thus, interactions can be found in both, the actor actions and in the system actions. From a black-box point of view, different semantics are applied to inputs and outputs. Interactions that are assigned to the system are called *SystemAssertions*. Interactions between the user and the system are subsequently treated as *ActorInteraction*. Whether a subtask contains an actor interaction or system related interaction is immediately apparent from the type of action.

Figure 4.5 shows the ITERA meta-model for test modelling as an UML class diagram. The meta-model defines a modelling element `<<TestAction>>` as root element of the context model and a subclass of UML Action. This serves as an anchor point for the integration into the activity diagram. As direct subclasses of *TestAction*, the meta-model provides the stereotypes `<<SystemAction>>` and `<<ActorAction>>`. These help to distinguish between actions that can be influenced by the user and system actions that cannot be changed, but carry essential information on the expected behaviour of the SUT.

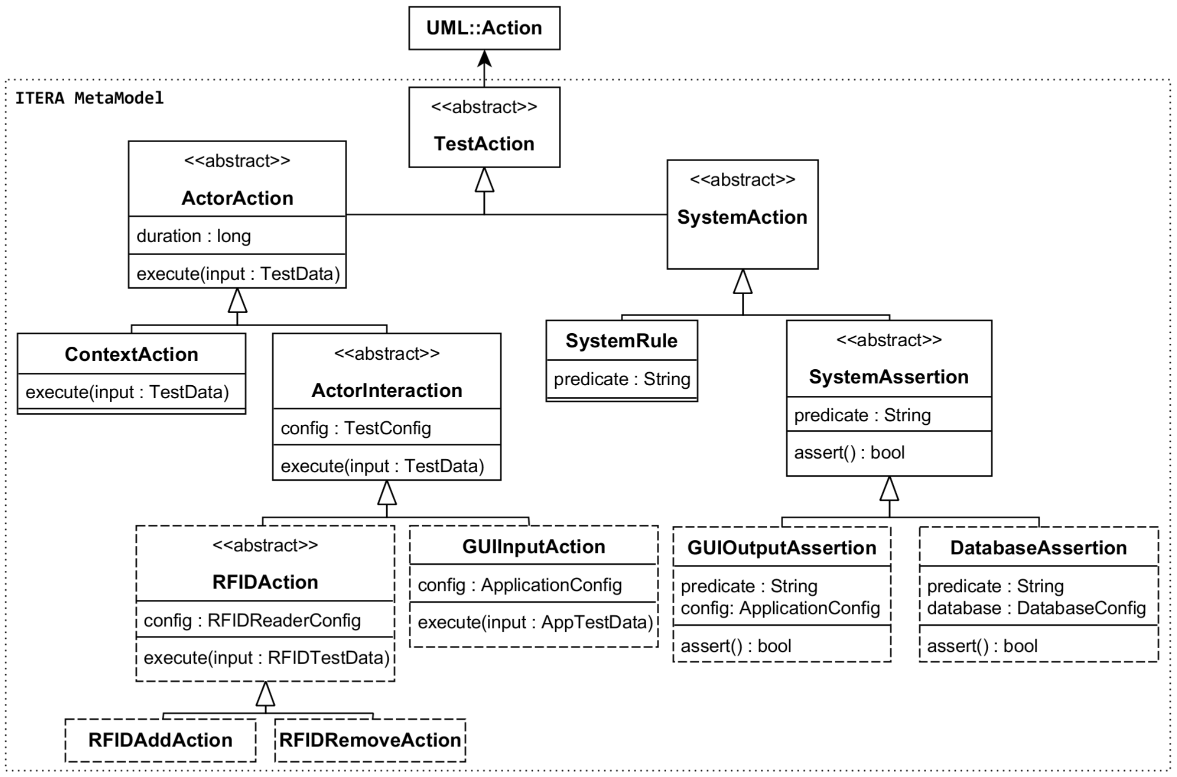


Figure 4.5: Class diagram of the ITERA Meta Model

ActorActions are further divided into `<<ActorInteraction>>` and `<<ContextAction>>`. Since each *ActorAction* describe a physical behaviour of a user all its subclasses are time-consuming. *ActorInteractions* reflect inputs for the SUT. *ContextActions* capture timing aspects, resulting from physical movements and processes where no direct interaction with the SUT is performed. *ActorInteractions* are strongly connected to the offered interfaces of the system and need to be extended by subclasses describing the different input types, namely `<<RFIDAction>>` (including `<<RFIDAddAction>>` and `<<RFIDRemoveAction>>`) and `<<GUIInputAction>>`. These are concrete implementations[1] of the meta-model and allow the stimulation of the SUT, by executing the contained `execute()` method. All classes shown with dotted lines need to be implemented in the test automation environment through appropriate technologies. When executing the tests, these are instantiated via reflections and their methods are executed. For a customisation of the ITERA method to support different application specific needs, the class `ActorInteraction` can be used as a possible extension-point. For example, this can be in form of a concrete implementation of a simulated light barrier or a weight scale.

SystemActions are divided into the subclasses `<<SystemRule>>`, reflecting alterations of the control flow based on bussiness rules, and `<<SystemAssertion>>` providing oracles for the evaluation of the test. Concrete implementations of *SystemAssertions* are `<<DatabaseAssertion>>`

[1]Except for RFIDAction, which is an abstract class and cannot be instantiated. However, the direct subclasses RFIDAddAction and RFIDRemoveAction realise RFIDAction.

and `<<GUIOutputAssertion>>`, both reflecting system specific mechanisms to evaluate the test. Similar to *ActorAction*, *SystemAction* offers another extension point for customisations, but also needs to be supported by the test execution framework through concrete implementations.

ActorAction

Actor actions are all activities in the use case, respectively in the activity diagram, which have a direct reference to actors or users of the system. They represent the behaviour of the user, the inputs made to the system and other activities that occur during processing of the use case. The superclass to model a user's behaviour is `ActorAction`. It incorporates the temporal aspects of the activities, such as the duration of the activity and is used to delay the test steps in the automatic test execution later on.

In accordance to the semantics introduced in Section 4.2 it is defined as:

Definition 4.3.1 (ActorAction)
An activity of type `<<ActorAction>>` is an action performed by an actor, which contains an attribute *duration* and results in an advance of time. Therefore, whenever an ActorAction is traversed (executed) the time is advanced.

ActorActions advance the time an action occurs at by a defined duration. This way the temporal aspects, which can influence the tag capturing mechanisms of RFID systems, originating from the physical processes surrounding the activities of a user are taken into account.

A distinction is made between interactions and context actions. Both are defined as subclasses of `ActorAction` and thus automatically inherit the timing aspects of its superclass.

ActorInteraction - *ActorInteractions* describe inputs made by an user via system interfaces. These include, for example, inserting an RFID tag into the reading field of an RFID reader, pressing a button via a touch display or triggering a light barrier.

Example: "When the user presses the button "finish" the process is completed." or "For this purpose, the user places the ID card in the designated compartment."

In cases where an interaction with a specific element or interface occurs, the corresponding model element must be able to uniquely identify the relevant element in the form of an interaction target element. In the meta model this is realized through the attributes of the appropriate classes, shown as *config*. For example the attribute `config` of a `RFIDAction` references a specific RFID reading device or a `GUIInputApplication` specifies the title name of a certain window of the operating system. If additional information (e.g. RFID tag) is also required, it must be possible to represent this information by means of a specific test input. In order to map such inputs in the test model, the subclasses use the parameters of the `execute()` method. The manipulation of certain inputs can be modelled by these parameters. However, a prerequisite is that the corresponding test automation can simulate the behaviours of these elements in form of a concrete implementation of a test driver.

The basic element for modelling RFID specific interactions is *RFIDAction*. It is further divided into *RFIDAddAction*, for placing RFID tags into a reader's read zone, and *RFIDRemoveAction*, removing these RFID tags again.

Analogous to the semantics of Section 4.2 a RFIDAction is defined as:

Definition 4.3.2 (RFIDAction)
An activity of type `<<RFIDAction>>` is an tuple *RFIDAction* with a reference to an RFID reader $r_n \in Readers_{sys}$ and a set of RFID tags $\{id_1, id_2, \ldots, id_n\} \supseteq Tags_{sys}$.

For example, when a user places an RFID tag inside the interrogation zone of an RFID reader, this is defined as an activity with the stereotype `<<RFIDAddAction>>` in the meta model. Akin to the set-based semantics of a RFID system a RFIDAddAction is defined as:

Definition 4.3.3 (RFIDAddAction)
An activity of type `<<RFIDAddAction>>` is defined as an action performed by an actor which results in a change of the tags present in a reader's sensing range. Therefore, whenever a RFIDAddAction is traversed (executed) it performs the operation:

$$reader_i(t_{n+1}) = reader_i(t_n) \cup \{id_1, id_2, \ldots, id_j\}$$

Similar, when a user removes an RFID tag from the interrogation zone of an RFID reader, it is described as an activity with the stereotype `<<RFIDRemoveAction>>`. It is defined in conformance of the semantics as:

Definition 4.3.4 (RFIDRemoveAction)
An activity of type `<<RFIDRemoveAction>>` is defined as an action performed by an actor which results in a change of the tags present in a reader's sensing range. Therefore, whenever a RFIDRemoveAction is traversed (executed) it performs the operation:

$$reader_i(t_n) = reader_i(t_{n+1}) \setminus \{id_1, id_2, \ldots, id_j\}$$

Contextual Actions - ContextActions are used to describe the external behaviour of the business process, which have no direct interaction with the application but are essential for the progress of the use case. For example, movements from one place to another or the selection of a certain book. ContextActions have in common that time advances during the physical process they describe, but no state changes of the SUT or the RFID system occurs. This property is an important aspect that has no semantic equivalent in comparative works, which use activity diagrams to generate test cases. Usually these approaches just model system functionality and user behaviours with direct influence on the SUT. However, for the evaluation of RFID applications timing aspects can play an important role, since many configuration settings, especially those for tag capturing are often time based.

Example: “The user selects some books from the library stock and goes to the lending terminal.”

In the meta model this is defined as an activity with the stereotype `<<ContextAction>>`. In terms of the semantics described in Section 4.2 it is defined as:

Definition 4.3.5 (ContextAction)
An activity of type `<<ContextAction>>` is defined as an action performed by an actor which does not change any of the tags present in a reader’s sensing range. However, it advances the time.

$$\forall r_i \in Reader_{sys} : reader_i(t_n) = reader_i(t_{n+1})$$

Within the test execution framework (Section 5.3), reflections are used to create instances of the stereotyped activities (UML classes implementing the functionality of the meta model). For each activity contained in the activity diagram, which refines `<<ActorAction>>`, the parameter duration is used to execute a sleep function, e.g. `System.sleep(duration)`. No other modification of the system are performed, thus, the `<<ContextAction>>` just contains an empty `execute()` method as it only consumes time.

SystemAction

In contrast to the ActorActions, SystemActions directly correspond to the requirements of the specification that are implemented in the system under test. Usually they are described through (business-)rules, which are supported by the respective application. In the context of the ITERA test methodology SystemActions are used to map the functional requirements of the system in the test model, used to select appropriate test data and to evaluate the test results. A distinction is made between *SystemRule* and *SystemAssertion*.

SystemRules - The control flow of an application is significantly determined by decisions of the user or by the SUT. Unlike the arbitrary decisions of a user, these are defined by the requirements, in particular by business rules. SystemRules influence the control flow of a RFID application from the systems perspective. This can be, for example a check for valid entries or based on certain system statuses. Even though they define logical conclusions, they do not directly address the specific computations in the final implementation of the SUT. Each of the conditions (guard conditions in the activity diagram), under which a guarded transfer from one activity to another can occur is, thus, specified by a corresponding SystemRule.

SystemRules in the context of the ITERA meta model contain a proposition that defines or limits an aspect of an operation and are used to define the structure or influence or control the behaviour of the system. In the meta model this is defined as an activity with the stereotype `<<SystemRule>>`.

Definition 4.3.6 (SystemRule)
An activity of type `<<SystemRule>>` is defined as an action performed by the system which includes a decision and is used as `decisionInputFlow` for the associated UML::DecissionNode. The outcome of the SystemRule affects the control flow of the activity diagram. It contains a predicate to specify an application's state.

The definition of an SystemRule is in conformance to the specification of activity diagram where the DecisionNode's guard condition is directly influenced by an preceding activity. (As a side-note a UML "Behavior" can be an Activity.)

> "If a DecisionNode has a decisionInput, then this must be a Behavior with a return Parameter and no other output Parameters. This Behavior is invoked for each incoming (control or object) token, and the result returned from the Behavior is available in the evaluation of the guards on outgoing edges." [47, p. 390]

> "For use only with DecisionNodes, a predefined guard "else" (represented as an Expression with "else" as its operator and no operands) may be used for at most one outgoing edge. This guard evaluates to true only if the token is not accepted by any other outgoing edge from the DecisionNode." [47, p. 390]

A template for a SystemRule is then

```
IF [predictate]
THEN [guard condition]
ELSE ([guard condition])
```

The modeller of a activity diagram may be advised to consider this first:

> "In order to avoid non-deterministic behaviour, the modeler should arrange that at most one guard evaluate to true for each incoming token. If it can be ensured that only one guard will evaluate to true, a conforming implementation is not required to evaluate the guards on all outgoing edges once one has been found to evaluate to true." [47, p. 390]

Using the previously introduced set-based semantics of RFID applications introduced in Section 4.2.4 the *SystemRules* can often be directly translated. Some examples for a mapping of SystemRules to executable semantics from the running example SUT Library are presented below:

SystemRule "check account status"

```
IF "[user] exists AND [user].status = valid"
THEN "account valid"
ELSE "account not invalid"
```

In the semantics of the set-based RFID application model the predicate of the SystemRule "check account status" can be described as:

$$user \in \mathit{ValidUsers}$$

$$\mathit{ValidUsers} := \sigma_{status=[''valid'']}(\mathit{Users})$$

The application of the selection operation σ defined by the relational algebra [121, 122] on a test database returns the set of valid users *ValidUsers*, which fulfil the selection criteria `\status=valid"`. In terms of testing, any user (e.g. an identifier of a RFID tag corresponding to this *user*) of this set represents a valid input in order to execute the path with the guard condition `\user valid"`.

System Rule "check user authorization"

```
IF "[user].rentals < 4 AND [user].invoice = 0"
THEN "user authorization approved"
ELSE "user not authorized"
```

In the semantics of the set-based RFID application model the System Rule "check user authorization" can be described as predicates.

$$\mid \sigma_{user=[user]}(\mathit{Rentals}) \mid < 4 \wedge \mid \sigma_{user=[user]}(\mathit{Invoices}) \mid = 0$$

The applying the selection operations on a test database a set of users is returned which fulfil the criteria where each *user* has less than 4 books currently rented and no open invoices.

SystemRule "check book authorization"

```
IF "[book].status = rentable OR
([book].status = reserved AND [book].reservation.user = [user])"
THEN "book rentable"
ELSE "book not rentable"
```

In the semantics of the set-based RFID application model the System Rule "check book authorization" can also be described as predicates[2]:

$$book \in \mathit{Rentable} \vee (book \in \mathit{Reserved} \wedge (user, book) \in \mathit{Reservations})$$

$$\mathit{Rentable} := \pi_{[book]}(\mathit{Library}) \setminus (\pi_{[book]}(\mathit{Rentals}) \cup \pi_{[book]}(\mathit{Reserved}) \cup \pi_{[book]}(\mathit{Reference}))$$

[2]The projection π is a operation defined by relational algebra [121].

SystemAssertions - Activities of type *SystemAssertion* describe the externally visible behaviour of the system. As they also reflect the corresponding system output, in the ITERA context they are used as assertions to ensure that the system has acted correctly. Similar to *ActorActions*, the system can offer several interfaces for communication with the environment. Therefore, an extension of the stereotype by application specific test drivers is possible, but only under the assumption that appropriate changes have been made to the following steps of the test methodology.

Typical examples are checking for certain elements of the user interface for specification conformity, e.g. comparing the contents of a message dialogue text field with an oracle. Another possible application is the comparison of the current screen content with a template (screenshot) created in advance. But they can also be used for the verification of certain properties, such as the status of an RFID tag in the database.

Example: "If borrow-ability was successfully evaluated, the rental object is marked as borrowed in the library system."

In the meta model this is defined as an activity with the stereotype `<<SystemAssertion>>`.

Definition 4.3.7 (SystemAssertion)
An activity of type `<<SystemAssertion>>` is defined as an action performed by the system which contains a assertion (predicate) and is used to specify an output.

Assertions in the context of the *Integrative Test Methodology for RFID Applications* (ITERA) test method help to determine a test input in the following step.

SystemAssertion "mark book rented"

```
[book].status = rented AND [book].rentals.user = [user]
```

In the semantics of the set-based RFID application model introduced the SystemAssertion "mark book rented" can be described as:

$$(user, book) \in Rentals$$

4.3.3 ITERA Domain Model

The domain model is intended to identify test data in the third step of the ITERA method, test case generation and the test data determination.

During the analysis of the specification, a UML class model is developed. This model serves as a *DomainModel* inside the context of the ITERA test method. Its purpose is to identify domain terms like `RFIDTag` and the objects this RFID tag is associated with. In Figure 4.6 both `Book` and `User` have an association to an RFID tag. Hence, these classes can be used as

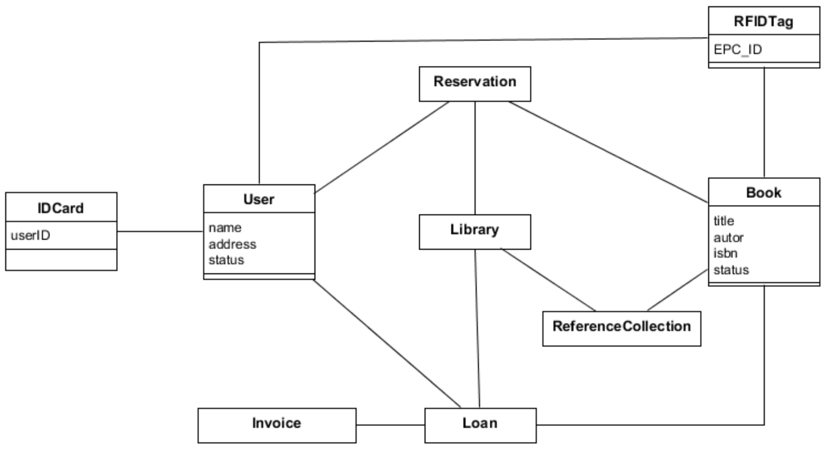


Figure 4.6: ITERA Domain Model of the example library

input parameters for the stereotyped actions `<<RFIDAddAction>>` and `<<RFIDRemoveAction>>`. Furthermore, the domain model is also used to refine the semantics of `<<SystemRule>>` and `<<SystemAssertions>>`. For example a book was rented ($book \in Rented$) by an user, can be described as shown in 4.2.4 using relational algebra.

$$book \in Rented \wedge (user, book) \in Rentals$$

where *book* is the book rented and user is the *user* which this was rented to. Using this notation valid input data for testing could be selected automatically by fulfilling these predicates from a previously created test database using a transformation into a domain specific language, e.g. a SQL statement.

4.3.4 ITERA Physical Model

RFID applications rely on the timing of the RFID events, thus, one important aspect for creating tests for RFID applications is the timing behaviour. The activity diagram can already capture the movement of RFID tagged objects but significantly lacks timing. Therefore the physical model is used to define these. There are different ways to determine the duration of activities.

1. Rough time estimation
2. Empirical Time Measurement
3. Simulation of Business Processes (Discrete Event Simulation)
4. Methods Time Measurement (MTM)

However, it depends on the use case which of the afore mentioned methods should be used. In general each method has it's advantages and draw-backs. Rough time estimation is easy to perform, but might result in a divergence to the real world values. In contrast to this Empirical Time Measurement might be more precise, but is also associated with a bigger invest and sometimes difficult to perform, as the environment must already prepared with RFID readers and tags. Building a Discrete Event Simulation does not require the RFID solution already be deployed, but might be inappropriate in comparison to the effort developing it. Contrary to Methods Time Measurement where the biggest hurdle might be the training needed to apply the method.

As it can be seen, there is no perfect solution that fits all needs, but the decision which method to choose, should be based on the question: “How critical are timing behaviours?” in the concrete use-case. If there is a significant time dependency, methods that result in accurate timing behaviours should be preferred over those with minimal effort in the survey phase.

Rough time Estimation - Discrete points of time are estimated roughly by the tester and used to refine the ActorActions. This might already be sufficient for processes where timing can easily be estimated. However, in some circumstances where the overall process time and the reading polling interval of an RFID reader is quite similar problems may arise as this method might be too general.

Empirical Time Estimation - Measurement of the process times can also be achieved by using a stop watch. By repeated measurement, the average duration of an activity can be determined. The intervals that are determined in this way are usually more accurate than by a mere estimate. However, this presupposes that the process to be analysed already exists in this form and that a measurement can be performed.

Simulation of Business Processes - Often *Discrete Event Simulation* (DES) is used to predict the outcome of changes in business processes. The same applies for the timing of these processes. A time estimation via DES can be realised using a physical model of the system’s environment. Figure 4.7 shows a prototype of a physical model, which was used in order to determine the

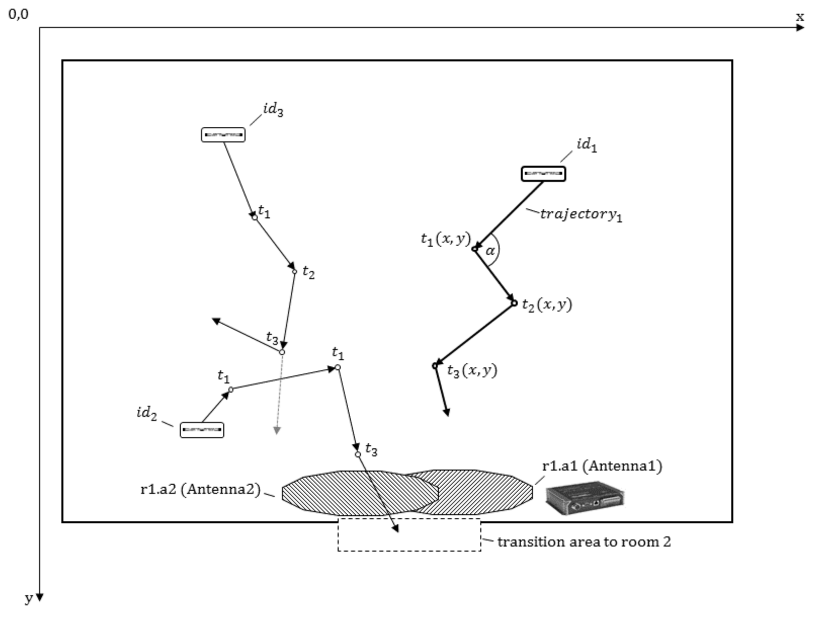


Figure 4.7: Simulation model for determination of time between RFIDActions

duration of the processes in a warehouse. Here a mapping of the real world environment into a the coordinate system of the model has been used to reflect distances. Each RFID tag has a movement operator which determines the direction and the speed. At each step intersection tests are performed. Once a RFID tag intersects with the polygon of a RFID reader's antenna a RFID Event and the duration of the movement can be recorded.

Methods Time Measurement - *Methods Time Measurement* (MTM) [125] is a system of predetermined motion times, which captures the basic movements of humans when performing tasks.

When MTM is used, all movements performed by humans are traced back to certain basic movements for which the required time is known. The smallest movement elements are recorded in MTM-1, which divides the work process into movement patterns such as "reach", "grasp", "release", "apply pressure", "disengage" etc. supplemented by movement elements such as "walking", "turn", "visual control" and so on.

In addition to the aggregation of motion elements, these procedures systematically compress data in such a way that, on the one hand, the double-hand analysis practised in MTM-1 (left and right hand are analysed separately according to fixed rules) and, on the other hand, the accuracy of parameters such as gripping distance and joining accuracy can be reduced to a minimum.

With these higher-level methods, the duration of more complex workflows can be determined relatively quickly. Due to the larger and perhaps not always exactly suitable modules, the accuracy of the system is only sufficiently statistically secured if a certain analysis scope is reached. This results from the compensation of the individual errors according to the Gaussian Law on Error Compensation, according to which the total error is lower than the sum of the individual errors.

4.4 Application of the ITERA Methodology

After the definition of the semantic model in Section 4.2 and introduction of the ITERA meta model in the previous section (4.3), a concrete example application of the ITERA methodology is given in the following subsections. Analogous to the four phases defined in the ITERA Methodology (see Section 4.1.2), first a use case from the specification will be transferred into a activity diagram in Section 4.4.1. Subsequently in Section 4.4.2, this activity diagram will be refined by the stereotypes defined in the ITERA meta model. In the next section the test model obtained this way will then be used to extract test cases.

4.4.1 From use case to activity diagram

The first phase (see Section 4.1.1) of the ITERA methodology is the systematic creation of an activity diagram derived from the specification. First of all, use cases are determined on the basis of use case diagrams. The aim is to identify RFID relevant use cases and convert them step-by-step into activity diagrams. In the following steps, this process is illustrated by example of a RFID equipped library.

It is assumed that requirement specifications are available, which informally reflect the requirements of the system to be tested or the system to be developed. These documents should identify a number of individual functional aspects, which at least partially describe the desired target behaviour of the system.

In the approach presented in this thesis, functional tests are carried out from a purely black-box perspective. Hence, the model is merely intended to describe the system's black-box behaviour as perceived by the system environment. Therefore, a specification that describes the functional and process-related requirements of the system, i.e. the behaviour perceptible by the users or the system environment is required. The resulting model should not describe details of the technical implementation, which are insignificant from the viewpoint of system tests.

Example Use Case

In a first step, it is necessary to identify the components that are functionally related to each other in the requirement documents. A use case diagram can be created to obtain an overview of the various functions that the system is going to implement. The content of the use case diagram is the desired, externally visible system behaviour. Thus it is a part of the requirements that the system is supposed to meet. In use case diagrams, however, the activities within the individual uses are not considered. The aim is to identify the core functionality of the RFID application and the relationships between the application and its actors.

Figure 4.8 shows the use case diagram created during requirements analysis. In this example, it contains eight use cases and two actors. Every actor in the diagram has an association to a use case when he takes over tasks in the particular use case.

Use cases where first described by Ivar Jacobson [123] in 1987 as "a special sequence of transactions, performed by a user and a system in a dialogue". Use cases often serve not only as a tool for designing the software, but can also serve as a basis for selecting test cases. In general, a use case contains steps and interactions of external actors with the system to perform a particular task. It is possible that an application may also contain alternative execution paths that are bound to certain conditions, as well as some exceptions where it is no longer possible to complete the task.

Use cases are often written in natural language and must therefore be refined in order to achieve detailed instrumentation for tests. For this reason, the initially informal descriptions of the

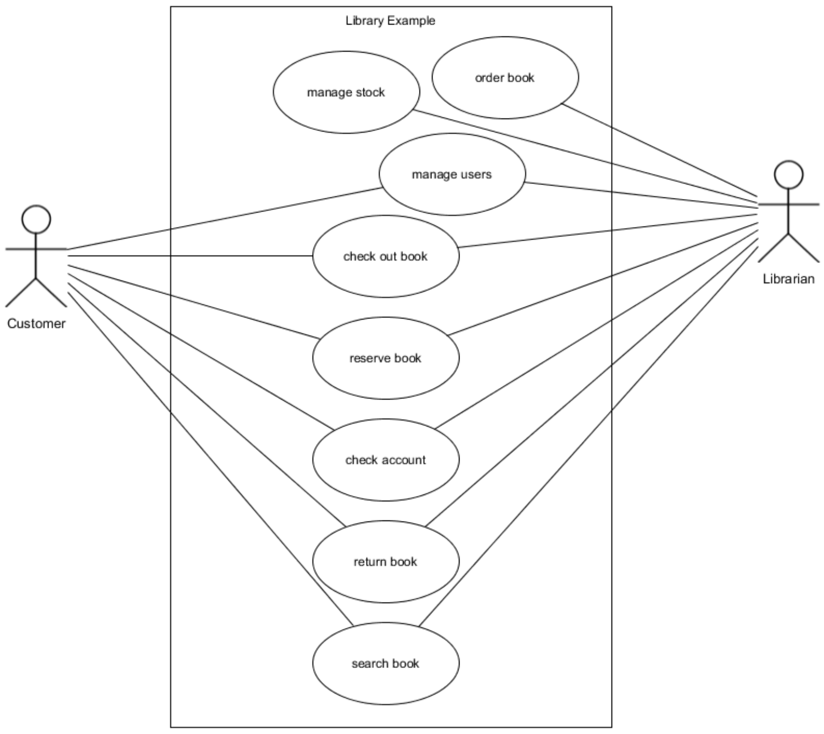


Figure 4.8: Example Use Case Diagram of a library

individual use cases are broken down by their functions in order to divide them into individual steps. Use cases can and should be continually adapted during discussions with stakeholders and by analysing other sources of demand. For simplification, the creation can be carried out in two steps: Initially, only the essential steps for a successful completion are covered. In a second step, these are then refined by adding exceptional cases.

Obviously, the most complete, detailed and consistent specifications of requirements at the beginning of a development project are desirable. In early phases of software development, however, it is not always possible to produce such precise documents - be it for lack of time or because of uncertainties about the exact design of the system to be developed. Therefore, the prerequisite that the original requirements documents must be complete or free of contradictions is waived. However, the activity diagram created in the following steps should be complete and consistent.

The table 4.2 shows the use case “check out books” of a sample library. According to Cockburn [124], use cases always contain a target that is contained in the description. The basic steps (also called *main success scenario*) describe the easiest way to achieve this goal. One step is an elementary action, which means that it cannot be broken down any further.

The pre-condition specifies the minimum requirements with which the described task can be started. The post-condition specifies possible effects or the result of the execution. Different conditions of can be linked with OR or AND to display possible results or exception handling during task processing.

Name	check out books
Description	The use case describes which checks are necessary in which order, so that a customer can borrow a book from the library. The outcome is that the user successfully rented a book.
actors	user, self-checkout terminal
pre-condition	the user has chosen some books to rent AND the self-checkout terminal is ready
post-condition	the book was marked as borrowed
basic steps	1. identify user 2. check user authorisation 3. identify rental object 4. check borrow-ability 5. save loan and mark book as rented 6. finish rental process
alternative steps	2.a identification failed (Exception / Condition) 3.a book could not be borrowed (Exception / Condition) 6.a user wants to borrow more than one book (Real alternative / Condition)

Table 4.2: Use Case: “check out books”

Since in practice individual requirements often do not completely satisfy the above pattern, an illustrative breakdown of the use case is shown. To illustrate this, the textual description of the “check out book” use case is used. In order to allow for a better understanding of the use case and to supplement the textual description, the self checkout terminal is shown in Figure 4.9.

Use Case “check out book”: “The user selects some books from the library stock and moves to the self checkout terminal. After the start of the rental process, the terminal requires the user’s identification with the library card (ID card). For this purpose, the user places the ID card in the designated compartment. The system then checks the user’s authorisation. If the authorisation was successful, the user is asked to place the books to be borrowed on the designated shelf space. A RFID reader embedded in the shelf then records the books. If the borrow-ability has been successfully evaluated, the book is marked as ”rented“ in the library system and the terminal display is updated accordingly. When the user presses the finish button a summary is displayed and the process is complete.”

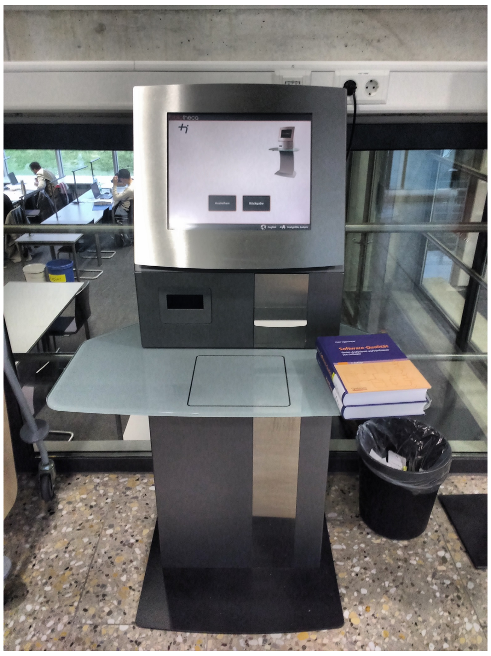


Figure 4.9: Self checkout terminal of the example library

Example Activity Diagram

For each use-case with relevance to the RFID system test, an activity diagram is created. Since it is often not possible to automatically transfer the use cases into activity diagrams, this step must be carried out manually. Figure 4.10 shows the derived activity diagram of the use case “check out books”.

It should be noted that when modelling process flows using activity diagrams, several test cases are modelled together for a given test problem. An activity diagram thus defines an entire test suite. For this reason, the individual test cases are initially not displayed in isolation, but in combination of different possible execution paths of the modelled application. Since the activity diagram is to be used as a basis for determining the test cases, it is necessary that the individual requirements are so fine-grained that a typical concrete execution scenario (sequence of context steps and interaction steps) is visible. If not, it may be necessary to further refine the requirements.

The activity diagram created this way should be lightweight and retain the natural language to be used as a communication medium between the requirements engineer and the customer. In order for the model to be suitable for the test purpose, the semantics of the individual activities must be clearly defined. Inputs and outputs should be clearly visible. User and system activities must be clearly separable. Alternative paths or their conditions must be explicitly listed and evaluable for the test.

The concept of model-based test automation for RFID application examined in this thesis is designed to generate added value through the systematic generation of specific test cases in

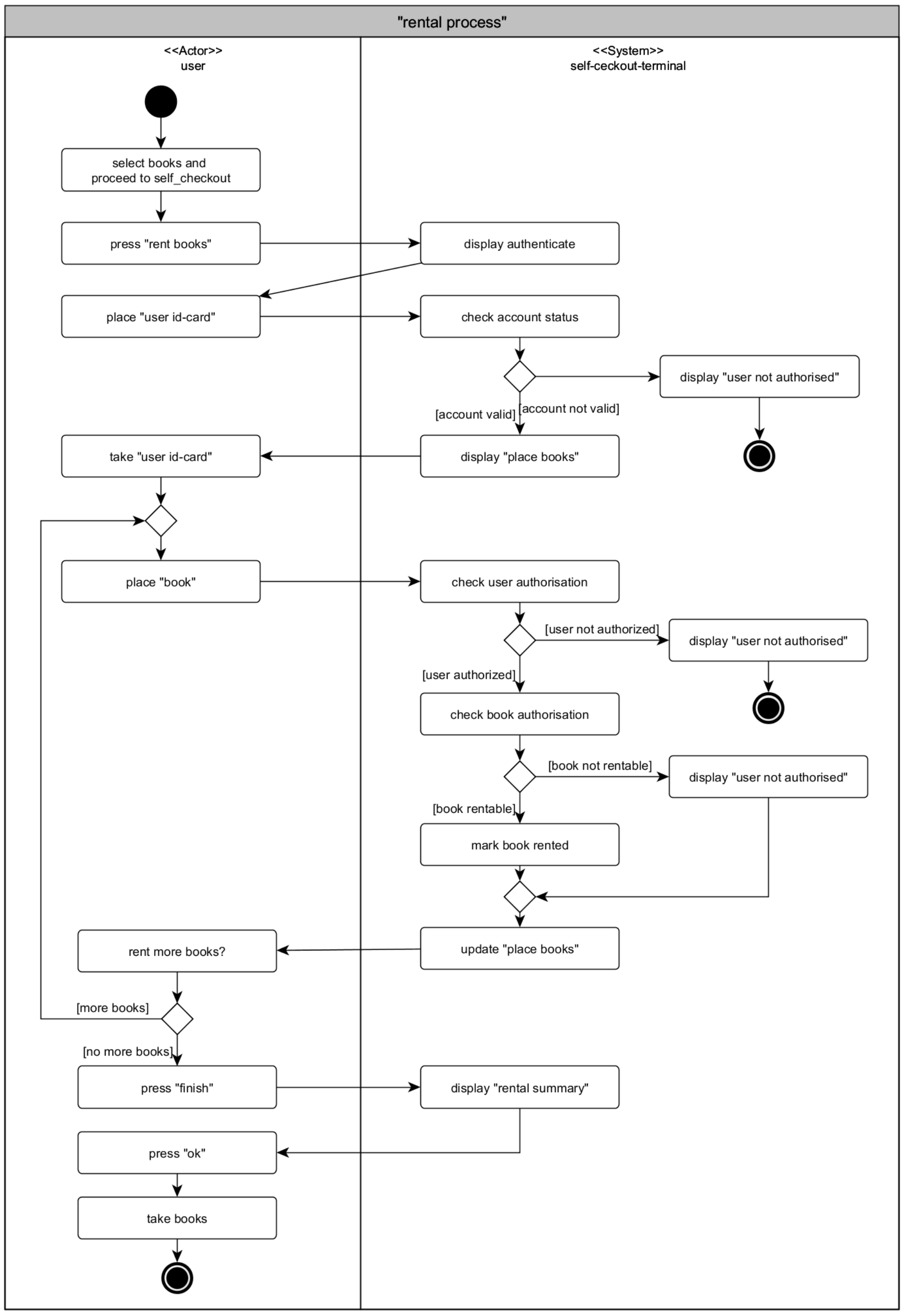


Figure 4.10: Example Activity Diagram derived from the use case refinement of “rental process”

software processes that do not force system modelling. Therefore, the test execution examined in Chapter 5 explicitly allows the manual creation of test cases.

4.4.2 Application of the ITERA Meta Model

In Section 4.3.2, the UML profile for test case modelling was introduced and the individual stereotypes have been discussed. Here the application of the UML profile is explained in detail using the previous example.

The second phase of the ITERA method (see Section 4.1.1) is concerned with the refinement of UML models with test-specific data before they are transformed to abstract test cases. This refinement manifests itself by specifying actions using elements of the Action type defined in the meta model.

Figure 4.11 shows a rudimentary activity diagram for the rental process of a RFID equipped self-checkout terminal in the running example library. The ITERA profile has been applied to the Activity Diagram and each activity has been stereotyped successively.

The first action (1) in the control flow of the activity diagram describes the user's actions, in particular the selection of books and a movement. There is no direct interaction with the system. However, it is important to note that time passes in each of these activities. Basically, these activities represent the flow of RFID tags through the business process. As already explained, timing events can have an influence on the capturing of RFID tags. Therefore, they are marked as `<<ContextActivity>>` in the test model. However, the actual flow of RFID tags does not have to be realized, as all RFID tags are globally available in the subsequent simulation environment.

Step 2 is the user's first interaction with the system. Here, the rental process is started by pressing the "rent books" button. As the user's interaction happens with the graphical user interface of the SUT it is marked with the stereotype `<<GUIInputAction>>`.

After the rental process has been started, the system prompts the user to identify themselves. Since all outputs of the system are test relevant, as they are part of the test evaluation process for black-box testing techniques, an assertion of type GUIOutput (`<<GUIOutputAssertion>>`) is used in step 3.

The authentication of the user (4) is carried out, by means of an user-id-card equipped with a RFID chip, so here the first interchange via a RFID reader takes place. The placement of an RFID tag in the reading zone of a RFID reader can be described by the stereotype `<<RFIDAddAction>>`.

The RFID tag read by the RFID system is then assigned to a user and the status of the user account is checked. Step 5 is therefore a system rule and can be specified by the type `<<SystemRule>>` as defined in the meta-model. Depending on the evaluation of this system rule, a case distinction is made. Either the user has a valid account and the system prompts the user to place the books to be borrowed on the corresponding area (7 - `<<GUIOutputAssociation>>`) or the transaction is acknowledged by displaying an error message (6 - `<<GUIOutputAssertion>>`).

In step 6, the user takes the user id-card, which is of type `<<RFIDRemoveAction>>`. As a side-note, a `<<RFIDAddAction>>` always has to be followed by an `<<RFIDRemoveAction>>` or

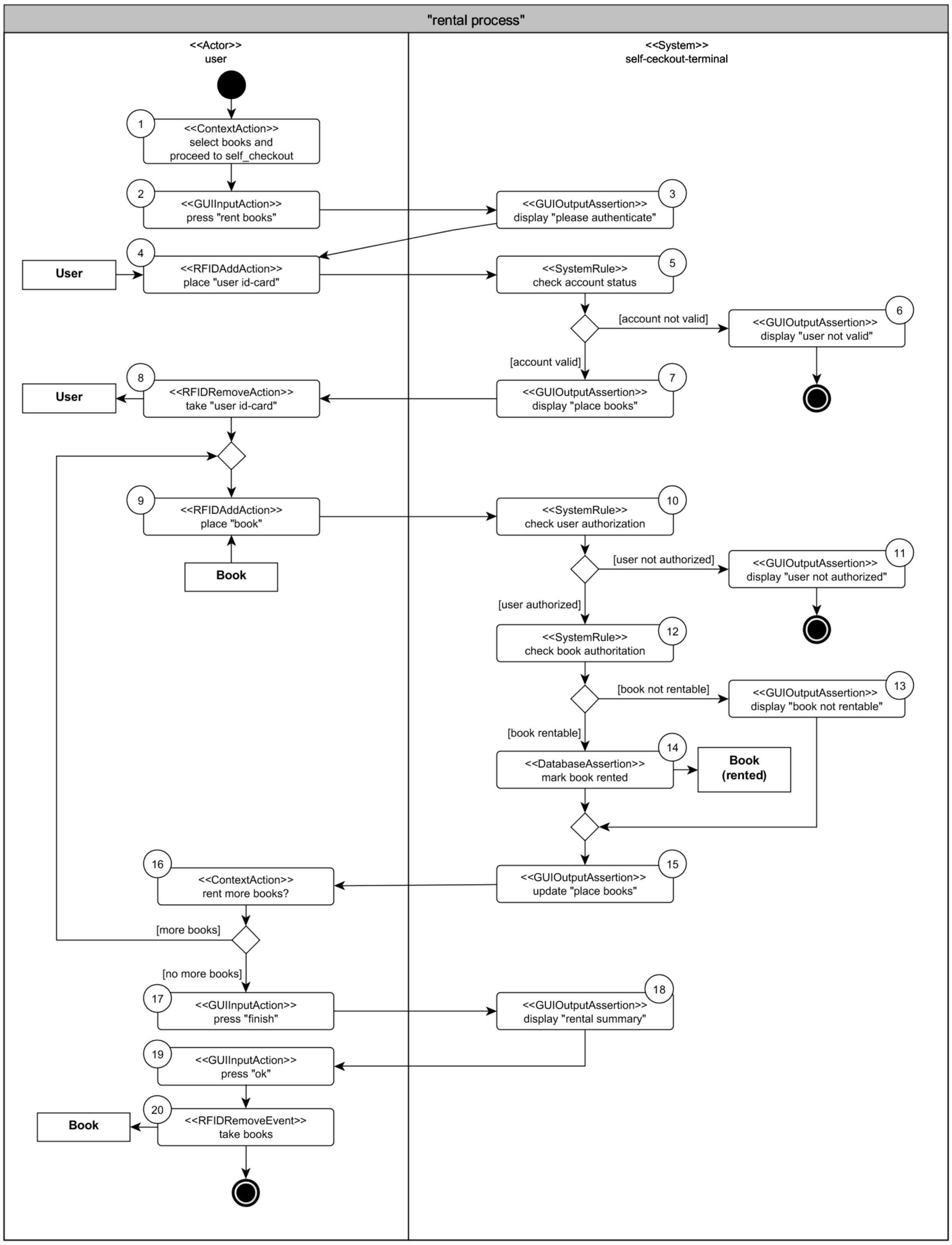

"rental process"
<<Actor>> user
<<System>> self-ceckout-terminal
1 <<ContextAction>> select books and proceed to self_checkout
2 <<GUIInputAction>> press "rent books"
3 <<GUIOutputAssertion>> display "please authenticate"
User
4 <<RFIDAddAction>> place "user id-card"
5 <<SystemRule>> check account status
[account not valid]
6 <<GUIOutputAssertion>> display "user not valid"
[account valid]
7 <<GUIOutputAssertion>> display "place books"
User
8 <<RFIDRemoveAction>> take "user id-card"
9 <<RFIDAddAction>> place "book"
Book
10 <<SystemRule>> check user authorization
[user not authorized]
11 <<GUIOutputAssertion>> display "user not authorized"
[user authorized]
12 <<SystemRule>> check book authoritation
[book not rentable]
13 <<GUIOutputAssertion>> display "book not rentable"
[book rentable]
14 <<DatabaseAssertion>> mark book rented
Book (rented)
15 <<GUIOutputAssertion>> update "place books"
16 <<ContextAction>> rent more books?
[more books]
[no more books]
17 <<GUIInputAction>> press "finish"
18 <<GUIOutputAssertion>> display "rental summary"
19 <<GUIInputAction>> press "ok"
Book
20 <<RFIDRemoveEvent>> take books

Figure 4.11: Application of the ITERA Profile on the example library "rental process"

an UML end node of type *FinalNode* (21,22), eventually. Once the user has been successfully identified, he can start borrowing books by placing the them on the designated area. The books are detected by an antenna of a RFID reader integrated in the shelf compartment. Hence, stereotype `<<RFIDAddAction>>` (9) is used for the test parametrization.

The detected books are now processed by the system. This is done by two checks (10,12). Two business rules are applied for validation: *check user authorisation* for ensuring the user has the right to borrow books and *check book authorisation* to ensure the book can be borrowed. Since these are system decisions, both are stereotyped by `<<SystemRule>>`. Depending on the outcome of each check, different paths are selected in the activity diagram.

In action 12, if the preceding checks were successful, the books are marked as borrowed and the corresponding entries are made in the library software database. Since the database can be used as an interface to check the test output, the type `<<DatabaseAssertion>>` is used here. The approach to testing RFID applications presented in this thesis presupposes the verifiability based on databases. This requirement is based on the fact that RFID applications can be used without user interactions. Therefore, a testability is only given if an appropriate interface (here the database) is available.

The successfully borrowed books are now also displayed via the output of the software at the input terminal (15). The use of the `<<GUIOutputAssertion>>` stereotype makes sense here as for the previous outputs.

The user now has the possibility to borrow further books (16), whether or not this is a decision that the user has to make now. Since it is a decision that is independent of the system, this action is marked as `<<ContextAction>>`.

The following steps are similar to the ones used before. Therefore, it should be obvious which stereotypes can be applied to these.

4.5 Automatic Test Case Generation

In the previous sections, the method for model-based test case specification was presented. For this purpose, a meta model has been introduced that allows test relevant aspects to be considered during the modelling of RFID specific test cases and to encapsulate test data in the activity diagram.

In this section the fundamental steps of the ITERA test methodology, that allow for the derivation of test cases are described. In particular how transformation is used to convert activity diagrams to a graph in a first step and how the test cases or potential execution paths of the activity diagrams are computed.

The central tool for transferring models between different abstraction levels is the model transformation, which uses specific transformation rules to map a source model to a target model. Model transformation is a process that removes or adapts elements of the source model or creates elements in the target model that might be aggregated from elements of several source models.

The aim of model transformation is to preserve those aspects of the system that are particularly relevant in the target model while irrelevant aspects are removed.

Test cases are ordered sequences of individual actions that are processed during the execution of tests. Since activity diagrams may contain cyclic and concurrent control flow structures, direct application of deep-first or breadth-first search algorithms is not suitable for calculating such ordered sequences of test-relevant actions. To avoid this problem, the approach to generate test cases from UML activity diagrams investigated in this thesis follows the path of modifying the breadth-first search algorithm so that valid paths are also generated from activity diagrams with concurrent and cyclic structures.

A particular challenge when testing software is to find suitable test cases. Test case determination should therefore be performed through a systematic and structured process. For this reason, this section introduces a procedure that allows to extract them directly from the activity diagram created in the previous section. In order to ensure adequate coverage of the RFID application by tests, different coverage criteria for activity diagrams are also defined in this section.

The procedure for determining the test case is carried out in three steps: At first, the UML activity diagram is transformed into an intermediate representation (activity graph). In a further step, a search algorithm is then applied in such a way that the previously defined coverage criterion is fulfilled. Subsequently, test cases are created based on these paths, including test parameters from the meta-model. The obtained test cases also provide the starting point for the test data determination in the next section.

4.5.1 ITERA Test Coverage Criteria

Coverage criteria (see 2.2.1) can be applied at different levels of abstraction. On a level of abstraction close to implementation, they are usually based on source code. Nodes of the control flow graph reference individual statements in the source code. At the level of system testing, tests are concerned with the testing of a software against its specification, without the knowledge of the source code (black box test). Accordingly, the nodes of the control flow graph reflect the activities of the user with the SUT. This allows the SUT to be tested from the user's perspective to see whether the SUT complies with the specification when using defined stimuli, e.g. RFID events.

A direct application of the existing test coverage criteria to system testing is not possible. System tests ensure that every function of the SUT that can be reached by the user is addressed at least by one test. However, this does not guarantee that every single statement in the source code is also checked by this set of tests. Consequently, statement coverage (C_0 coverage, analogous to C_1 and C_2-coverage), focused on statements in the source code of the application, cannot be used in its original meaning for system tests. In the context of this thesis, the activity diagrams created in the previous steps are interpreted as a control flow graph and build the basis for the coverage metric.

To define coverage criteria at the abstraction level of UML activity diagrams, UML activity diagrams are defined as:

Definition 4.5.1 (Activity Diagram)
A activity diagram is a defined as a tuple $AD = (N, C, n_I, N_F, F)$ where

- $N = \{n_1, n_2, \ldots, a_n\}$ is a finite set of activities of type `ActivityNodes`
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions of type `ControlNode`
- $C = \{c_1, c_2, \ldots, c_m\}$ is a finite set of guard conditions, where c_i is related to t_i and depends on n_i.
- $n_I \in N$ is the start node of the activity diagram of type `InitialNode`
- $N_F \subset N$ is a finite set of end nodes of the activity diagram of type `ActivityFinalNode`
- $F \subseteq \{N \times T\} \cup \{T \times N\}$ is the flow relation between the activities and transitions of type *ActivityEdge*

An activity diagram based on the plain meta-model of UML does not clearly show which nodes represent interactions with the user, depict outputs of the SUT or are context relevant. For this reason, the UML profile (Section 4.3.2) has been defined, providing stereotypes that allow for assigning corresponding attributes to elements of an activity diagram and include concrete semantics.

Within the scope of this thesis, the activities of the activity model are considered as elementary test steps. Therefore, the definition of the coverage criteria listed below refers to activities that are relevant to the execution of the test. Test relevant activities are in this sense direct subclasses of the meta-model defined in the previous section. These are in particular based on the stereotypes `<<SystemAction>>` or `<<ActorAction>>`, such as `RFIDAddAction` or `RFIDRemoveAction` (derived from `ActorInteraction`). Based on the definition of the Activity Diagram and in regard to elementary test steps five coverage criteria are defined:

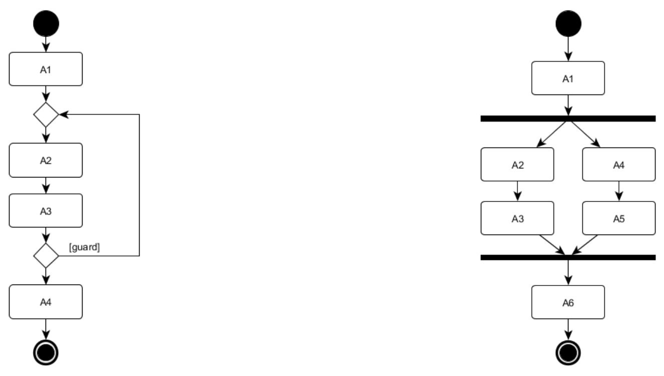
Definition 4.5.2 (activity coverage)
An activity coverage test (C_{0AD} test) for an activity diagram AD is defined as a test, during which all test-relevant activities are executed at least once. C_{0AD} tests are therefore based on the classical definition of C_0 tests, whereby the activity diagram of the SUT is used as the control flow graph.

Definition 4.5.3 (activity branch coverage)
A C_{1AD} branch coverage criterion for an activity diagram AD is defined as a coverage criterion that requires all test-relevant branch alternatives (i.e. outgoing edges to *DecisionNode* elements) in the activity diagram to be covered by at least one test case.

Definition 4.5.4 (activity path coverage)
Path coverage C_{2AD} criterion for an activity diagram is defined in analogy to the classical C_2 test as a test covering criterion, which requires that each path is covered by a test case in the activity diagram AD.

Analogous to the classic C_2 coverage criterion, however, the problem arises in practice that even for models with relatively low complexity, the number of possible paths quickly increases to a level that is not manageable even when using computers. This problem was identified in the literature as a state-explosion-problem (combinatorial explosion) (cf. Valmari [126], Peled [127]). The approach examined in this thesis avoids this problem by not considering the C_2 criterion, but the C_{2cAD} criterion discussed below.

Definition 4.5.5 (activity basis path coverage)
A C_{2cAD} baseline path coverage test for an activity diagram AD, is defined as a C_{1AD} test which, analogous to the classical C_{2c} test, executes each cycle in the activity diagram during test execution with at least a given number of passes, whereby concurrent control flow structures contained in the path are removed.



(a) cyclic structure (b) concurrent structure

Figure 4.12: Concurrent and cyclic structures of UML Activity Diagrams

The UML Activity Diagrams provide modelling elements that allow both cyclic and concurrent structures (see Figure 4.12). The classical definitions of the test coverage criteria are based on a control flow graph that does not contain concurrent structures. However, the activity diagram of the SUT is used as a control flow graph, in which concurrent structures are allowed. Whereas a test case intentionally does not include branch nodes, contain cyclic or concurrent structures to guarantee deterministic execution. In order to achieve this, the concurrent activities and loops must be interleaved when they are converted to test cases. If elements exist on concurrent control flow structures, only sequences of permitted permutations of these elements should be reflected by tests, assuming that a user can carry out exactly one interaction with the SUT at any given point in time.

Definition 4.5.6 (concurrent activity path coverage)
A Concurrent Activity Path C_{CAD} is defined as a C_{2cAD} (basis path coverage) test that passes through each cycle at least once and additionally tests permitted sequences of permutations of test-relevant elements on concurrent control flow structures. A sequencing of elements of concurrent control flow structures by permutation is permitted if elements modelled on the same control flow structure are not interchanged with elements of concurrent control flow structures in their execution sequence after permutation. To avoid the state explosion problem, the C_{CAD} criterion is met if at least one permissible sequencing is covered by a test.

The method examined in this thesis for generating system tests from activity diagrams implies that concurrent control flows in applications are described from the user's point of view and can thus represent sequences of adjacent individual actions. The rationale is that a human actor is not able to interact concurrently with multiple components of an application at the same time. Due to human capabilities, there is are always an implicit sequential ordering. However, when multiple actors perform the same activity, as it might be the case in e.g. a warehouse, this is not modelled in the activity diagram directly, rather this is covered during the execution of the tests by running multiple tests at the same time.

In order to take a closer look at the path explosion and to explain the interleaving of the actions, the diagram shown in Figure 4.12b can be considered. Assuming that $R \subseteq A \times A$ is the partial order relation of two activities $a_i \in A$ and $a_j \in A$. Precedence of activities, that is activity a_i has occurred before a_j, is referred to as $a_i \prec a_j$. The example shown in Figure 4.12b contains 4 concurrent activities and has the order relation $R = \{A_1 \prec A_2, A_1 \prec A_4, A_2 \prec A_3, A_4 \prec A_5, A_3 \prec A_6, A_5 \prec A_6\}$. This results in 24 permutations of which 6 are valid (permitted) activity paths (denoted P_i):

$$P_1 = A1 \rightarrow A2 \rightarrow A3 \rightarrow A4 \rightarrow A5 \rightarrow A6$$
$$P_2 = A1 \rightarrow A2 \rightarrow A4 \rightarrow A3 \rightarrow A5 \rightarrow A6$$
$$P_3 = A1 \rightarrow A2 \rightarrow A4 \rightarrow A5 \rightarrow A3 \rightarrow A6$$
$$P_4 = A1 \rightarrow A4 \rightarrow A5 \rightarrow A2 \rightarrow A3 \rightarrow A6$$
$$P_5 = A1 \rightarrow A4 \rightarrow A2 \rightarrow A3 \rightarrow A5 \rightarrow A6$$

Note that if these would been test cases, every single (test-)path ($P_1 - P_6$) already fulfils the C_{CAD} criterion, as only one valid sequence of the concurrent activities is required.

4.5.2 Transformation from an activity diagram to an ActivityGraph

In order to generate the test cases, the UML activity diagram is transformed into an activity graph in a first step. This graph is then traversed using an algorithm. The traces of the traversal form a set of valid sequences of actions that represent possible execution paths. By combining the resulting paths with data from the original activity diagram, the so-called abstract test cases are obtained which have to be enriched with test data by the tester.

To apply the algorithm for test case generation an *ActivityGraph* (AG) is created. The nodes and transitions of the AG directly correspond to those in the activity diagram, except that multiple UML FinalNodes are combined to one FinalNode in the activity graph. A modified *Breadth First Search* (BFS) algorithm is then used to traverse the graph and to obtain the test cases. This creates paths that represent the sequences of actions in the activity diagram. In the semantics of the activity diagram, each node visited means that the associated action is executed.

An AG is defined as shown below:

Definition 4.5.7 (Activity Graph)
An ActivityGraph $AG = (N, E)$ is defined as a directed graph, where

- $N = \{n_1, n_2, \ldots, n_n\}$ is a finite set of nodes and
- $E \subseteq \{(n_i, n_j) \mid n_i, n_j \in N, i \neq j\}$ is a finite set of ordered pairs, called directed edges.

Each node $n \in N$ has a structure (*nodeId*, *nodeType*, *nodeStereoType*, *nodeName*), where

- *nodeId* is an unique identifier attached,
- *nodeType* $= \{I, E, A, D, M, F, J\}$ as defined by the mapping rules (Table 4.3),
- *nodeStereoType* is the textual description of the stereotype of the activity and
- *nodeName* is a textual description of the node in correspondence to the activity.

During the transformation, an activity graph is generated from an activity diagram in accordance with the transformation rules defined below:

Transformation Rule 4.5.1 (UML Activity Node → ActivityGraph Node)
Let $AD = (N_{AD}, T, C, n_I, N_F, F)$ be a Activity Diagram and $AG = (N_{AG}, E)$ an ActivityGraph. During the transformation, each node $n \in N_{AD}$ creates a corresponding node $m \in N_{AG}$. If there is more than one node $n \in N_F$ of type ActivityFinalNode only one $m \in N_{AG}$ of type F is created.

Transformation Rule 4.5.2 (UML ActivityEdge → ActivityGraph Edge)
Let $AD = (N_{AD}, T, C, n_I, N_F, F)$ be a Activity Diagram and $AG = (N_{AG}, E)$ an ActivityGraph. During transformation each control flow $t \in T$ creates an directed edge $e \in E$ such that, $n_i \rightarrow n_j$ corresponds to an ordered pair (n_i, n_j) in the ActivityGraph.

Table 4.3 describes the mapping rules of the UML ActivityNode to the corresponding types in the activity graph.

An example of an AD-to-AG transformation is presented in Figure 4.13. It shows the application of the previously defined transformation rules on an example activity diagram. The transformation rule 4.5.1 transforms each of the individual elements of the UML activity diagram to nodes in the ActivityGraph (illustrated with dashed arrows). Additionally, each type of the UML ActivityNode is also mapped to the corresponding *nodeTypes* in the ActivityGraph in accordance

Rule	UML ActivityNode	Activity GraphNode
1	`InitialNode`	Node of type I
2	`ActivityFinalNode`	Node of type E
3	`ActivityNode`	Node of type A
4	`DecissionNode`	Node of type D
5	`MergeNode`	Node of type M
6	`ForkNode`	Node of type F
7	`JoinNode`	Node of type J

Table 4.3: Rules for mapping UML ActivityNodes to ActivityGraph nodes

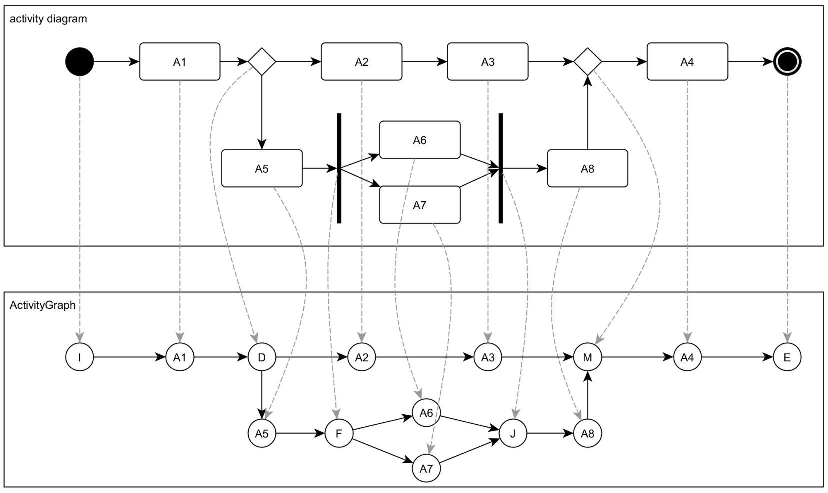


Figure 4.13: Example transformation of an activity diagram to an ActivityGraph

to Table 4.3. Transformation rule 4.5.2 created edges in the activity graph analogous to the control flow in the activity diagram.

The ActivityGraph shown in Figure 4.13 is a representation of the original activity diagram. If the graph is traversed, the transitions between the nodes correspond to the control flow in the activity diagram. Traversing a node is semantically equivalent to executing the corresponding activity. The graph created is the basis for the derivation of the test cases and thus reflects possible execution sequences of the modelled use case of the application.

4.5.3 Traversal of the Graph and Extraction of Test Cases

Test cases are ordered sequences of individual actions that are processed during the execution of tests. Since activity diagrams can contain cycles and concurrent control flow structures, a direct application of a plain search algorithm, like *Depth First Search* (DFS) or BFS, is not suitable for calculating such ordered sequences of individual actions. Therefore, the creation of execution paths takes place in several steps:

In a first step, the ActivityGraph is traversed by a modified breath-first search algorithm (listing 4.1), starting from node I, ending at node F and visiting each node in the graph exactly once.

In the semantics of the activity diagram, the resulting paths correspond to the execution of the corresponding activities. Therefore, execution paths generated in this way already fulfil C_{0AD}, since it is ensured that every action modelled in the activity diagram is included. If concurrent structures or cycles are discovered during traversal, either a new instance of the DFS or another version of the modified BFS (SequentialBFS, listing 4.2) is called.

Concurrent activities have to be interleaved, since they prevent the useful application of the original traversal algorithm (listing 4.1) and would result in impermissible test cases. The C_{ACD} criterion requires that only one reasonable permutation must be included, i. e. one valid sequence of concurrent activities, therefore the modified sequential breadth-first search algorithm (listing 4.2) can be utilised here.

However, the paths created this way do not yet contain cycles and do not necessarily end at an end point (ActivityFinalNode) of the control flow within the activity diagram. Therefore, the next step is to complete the generated execution sequences through cycles and then complete each path in such a way that it ends at an end point of the activity diagram. To achieve this, the BFS search is called recursively. Contrary to the initial call, this time the start node is the beginning of the cycle, the end node remains F. To control the path generation and to provide a termination condition a additional argument *depth* is used. This not only allows to avoid test path explosion, but also enables the tester to define boundaries by controlling how often cycles are traversed. A depth of 1 means cycles are traversed at least once, correspondingly a depth of 2 means each cycle is taken twice. Afterwards the supplemented paths fulfil both the C_{2cAD} and the C_{CAD} coverage criterion.

Activity diagrams can be hierarchically structured using the UML Call-BehaviorAction element, so that subordinate activity diagrams are referenced from parent ones. To generate test cases from activity diagrams, such dependencies must be taken into account during test case generation. Hence, in a final step, these dependencies are resolved by inserting the corresponding execution paths of the referenced activity diagram at these locations. This way, the complexity of determining test paths is kept low. This can otherwise be problematic especially in very large activity diagrams.

Like in comparative test case generation algorithms based on UML activity diagrams [66], a restriction on the model is needed. Nested control structures (such as DecisionNode and MergeNode) are not allowed in between concurrent sections. Here the modeller has to create a subordinate activity diagram containing the nested activities. The test paths of this subordinate activity diagram will be substituted at the corresponding places (original activity) similar to UML Call-BehaviorAction. However, in comparison to [66] nested cycles can be handled properly by the proposed algorithm (see Section 6.3).

The generated paths from the ActivityGraph represent different execution paths of the modelled use-case in the activity diagram, each of these paths represent a future test case.

Listing 4.1 shows modified algorithm used during traversal of the ActivityGraph:

Listing 4.1: ITERA Modified BFS (normal mode)

```python
def bfs(graph, start, goal, depth):
    paths = []
    queue = [(start, [start])]
    while queue:
        (vertex, path) = queue.pop(0)
        if nodeType[vertex] == goal:
            paths.append(path)
            continue
        if nodeType[vertex] == "FN":
            # Concurrent section detected
            seq_path = sequencer_bfs(graph,vertex,[goal,"JN"])
            last_node = seq_path[-1]
            for next_vertex in graph[last_node]:
                queue.append((next_vertex, path + seq_path + [next_vertex]))
                continue
        for next_vertex in graph[vertex]:
            if next_vertex in path:
            ## Cycle detected
                if depth > 0:
                    for cpath in bfs(graph,next_vertex,goal,depth-1):
                        paths.append(path + cpath)
        else:
            queue.append((next_vertex, path + [next_vertex]))

    return paths
```

Listing 4.2 is referred to as sequential BFS and mainly used for traversing concurrent structures inside the activity diagram. It differs in the way it keeps track of visited nodes.

Listing 4.2: ITERA Modified BFS (concurrent sections)

```python
def sequencer_bfs(graph,start,goal):
    queue = [start]
    visited = []
    while queue:
        vertex = queue.pop(0)
        for next_vertex in graph[vertex]:
            if nodeType[next_vertex] in goal:
                continue
            if nodeType[next_vertex] == 'FN':
                visited.append(next_vertex)
                path = sequencer_bfs(graph,next_vertex,goal)
                last_node = path[-1]
                visited.extend(path)
                for next_vertex in graph[last_node]:
                    path.append(next_vertex)
                continue
            visited.append(next_vertex)
            queue.append(next_vertex)
    return visited
```

Figure 4.14 shows the ActivityGraph of the continuously used example library “rental process” (shown in Figure 4.11) after the application of the mapping rules defined in Table 4.3. For comprehensibility the numbers of the activities have been conserved to make it easier to follow the transformation. Every activity within the activity diagram annotated with the stereotype `<<ActorAction>>` has been denoted **A**, respectively the activities of type `<<SystemAction>>` are denoted with **S**. Merge nodes (**M**), decision nodes (**D**) and the `InitialNode` (**I**) have been denoted following mapping rules. In accordance to the transformation rule 4.5.1 the nodes of

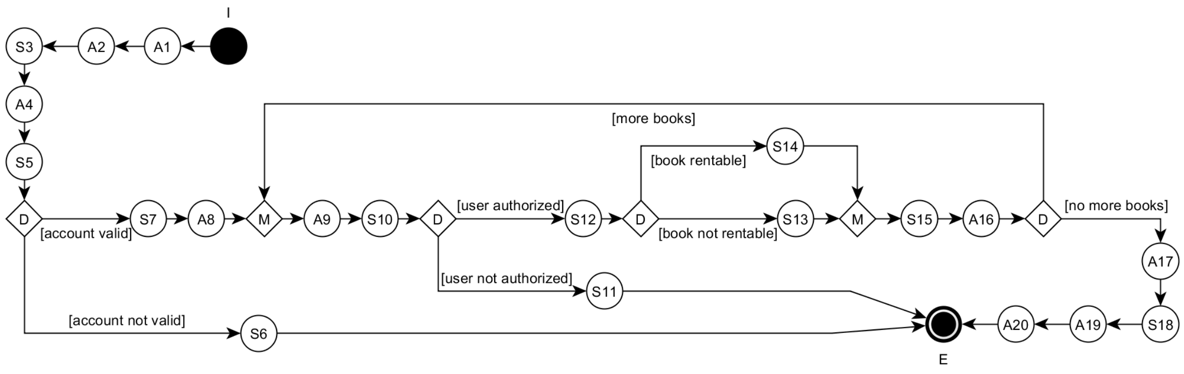


Figure 4.14: ActivityGraph of the example “rental process”

type `ActivityFinalNode` of the activity diagram have been combined to one node (denoted **E**).

The application of the proposed test case generation algorithm on the ActivityGraph “rental process” yields six test cases. The complete path of of Test Case 3 including the conditions (guard conditions at the UML Decision Nodes) of the corresponding decisions from the library example is shown below:
TC3: I → A1 → A2 → S3 → A4 → S5 → D[account valid] → S7 → A8 → M → A9 → S10 → D[user authorized] → S12 → D[book rentable] → S14 → M → S15 → A16 → D[no more books] → A17 → S18 → A19, → A20 → E

Due to the length of the individual sequences, the generated test paths are not completely depicted here. For simplicity, only the decision nodes and their conditions during the traversal are shown below:

TC	Conditions
1	[account not valid]
2	[account valid] → [user not authorized]
3	[account valid] → [user authorized] → [book rentable] → [no more books]
4	[account valid] → [user authorized] → [book not rentable] → [no more books]
5	[account valid] → [user authorized] → [book rentable] → [more books] → [user authorized] → [book rentable] → [no more books]
6	[account valid] → [user authorized] → [book not rentable] → [more books] → [user authorized] → [book rentable] → [no more books]

Table 4.4: Shortened test paths generated during traversal of the ActivityGraph of the “rental process”

There are two cycles in the used example “rental process”. As it can be seen in Table 4.4 cyclic paths, both zero loop and loop once, are contained in the generated paths:

- loop zero time:

  TC5: $I \rightarrow A_{(\text{place user id-card})} \rightarrow D_{[\text{account valid}]} \rightarrow A_{(\text{place book})} \rightarrow D_{[\text{user authorized}]} \rightarrow D_{[\text{book rentable}]} \rightarrow$

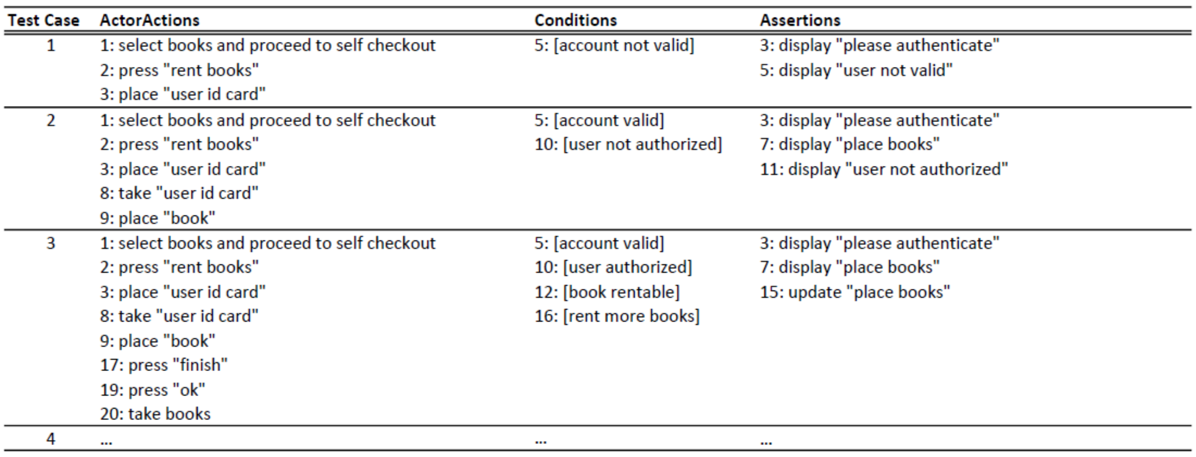
$D_{\text{[no more books]}} \rightarrow E$

- loop once:

  TC6: $I \rightarrow A_{\text{(place user id-card)}} \rightarrow D_{\text{[account valid]}} \rightarrow A_{\text{(place book)}} \rightarrow D_{\text{[user authorized]}} \rightarrow D_{\text{[book rentable]}} \rightarrow A_{\text{(place book)}} \rightarrow D_{\text{[user authorized]}} \rightarrow D_{\text{[book rentable]}} \rightarrow D_{\text{[no more books]}} \rightarrow E$

Since there are no concurrent structures in the example and each loop is run through at least once, the test cases generated fulfill the coverage criterion C_{ACD} defined in the previous section.

The resulting paths are now processed into abstract test cases and serve the tester to determine the test data. For this purpose, additional information from the classes of the meta model and of the individual nodes of the ActivityGraph are analysed and presented in a table. Figure 4.15 shows an excerpt of the prepared test cases of the used example library "rental process".



Test Case	ActorActions	Conditions	Assertions
1	1: select books and proceed to self checkout 2: press "rent books" 3: place "user id card"	5: [account not valid]	3: display "please authenticate" 5: display "user not valid"
2	1: select books and proceed to self checkout 2: press "rent books" 3: place "user id card" 8: take "user id card" 9: place "book"	5: [account valid] 10: [user not authorized]	3: display "please authenticate" 7: display "place books" 11: display "user not authorized"
3	1: select books and proceed to self checkout 2: press "rent books" 3: place "user id card" 8: take "user id card" 9: place "book" 17: press "finish" 19: press "ok" 20: take books	5: [account valid] 10: [user authorized] 12: [book rentable] 16: [rent more books]	3: display "please authenticate" 7: display "place books" 15: update "place books"
4	...	...	...

Figure 4.15: Excerpt from generated test cases

4.6 Test Data Determination for System Testing

When testing software systems, it is necessary to supply the SUT with input data. Since testing is not only intended to mimic a productive use of the system, special demands are imposed on the test data used. While test data should adequately represent the average (or typical) use case, it is also important to stimulate the SUT with input data that does not represent any typical use case. This includes, for example, data that is very close to or even exceeds the limits of validity. The selection of test data is therefore a demanding task.

The method presented in this thesis generates test cases based on a previously defined coverage criterion out of the control flow of the refined activity diagrams. Although some requirements for the selection of test data are already given by the generated test cases, these should be systematically selected as well. Automatic test input data selection seems therefore promising to even further assist the tester.

4.6.1 Automatic Test Data Derivation

The primary goal of automated test data generation is to reduce the time required to manually create the test data. For this reason, it is also useful to automatically generate test data for RFID application tests. Especially for this type of software, the complexity of possible input parameters quickly becomes unmanageable for a human actor (e.g. inherent state of objects with attached RFID tags, continuous read events, timing and physical challenges). To reach that goal, the question is whether the complexity of human actors can be managed and if the test input space can be divided such a way that meaningful representatives can be selected, so that a satisfactory solution can be achieved in terms of test quality. However, the necessary prerequisites for a systematic generation of test data for RFID applications are not fulfilled. This is partly due to the fact that input data, in this case RFID tags, only contain a minor part of the system's data, such as the identifier of an RFID tag. The identifier only serves as a reference to an existing data record of the application.

This can easily be seen by an example from the library application used in the previous sections. E.g. the book titled "Test Driven Development: by Example" by Kent Beck contains extensive entries in the database such as: TagID, title, author, ISBN, publisher, year of publication, etc. Even if this information (book title, author, etc.) would be reflected by the test model, it would still not be sufficient to automatically determine systematic test input data. Objects stored in the database used as input might also have complex relationships with other entries or hidden states. As an example, a certain book could have been borrowed or reserved by a specific user. Hence, the test model would also be suitable for capturing these relationships between the different data sets of the application.

4.6.2 Test Data Derivation

The identifier of an RFID tag often only serves as reference to an already existing data record. Therefore, the approach for testing RFID applications presented in this thesis assumes that a corresponding database with test data already exists. If not, a suitable database needs to be created by the tester first. To create such a database, data can either be pulled from production systems or generated by database generators. Examples for automatic generation of database populations can be found in various professional tool suites for testing.

Similar to comparable model-based test approaches, the assignment of test data in the ITERA test method is a manual process, too. However, a certain path generated from the activity diagrams can only be executed if the input data fulfils the criteria defined by the guard conditions along the traversed path. These can be used as the starting point for choosing valid test data from the test database. For example the generated test case 3 (see Section 4.5.3) contains two RFIDAddActions in addition to the GUIInputActions. The input data thus requires two RFID tags and corresponding identifiers. The conditions from test case 3 shown in Figure 4.15 are listed below:

1. Condition: [account valid]
2. Condition: [user authorized]
3. Condition: [book rentable]
4. Condition: [no more books]

Including the predicates from the associated test steps of stereotype `<<SystemRule>>` assigned to these conditions, the following (business-)rules are obtained:

- SystemRule for [account valid]:
  `(user.exits) AND (user.status = valid)`
- SystemRule for [user authorized]:
  `(user.rentals < 4) AND (user.invoice = 0)`
- SystemRule for [book rentable]:
  `(book.status = borrowable) OR`
  `((book.status = reserved) AND (book.reservation.user = user))`

Condition 4 “no more books” does not contain a associated rule, since it is a decision the user of the application made. Nonetheless, considering the rules and the conditions it should be obvious which explicit entries the tester needs to select as test input from the test database in order to allow the execution of the appropriate test case (here test case 3 from the running example). E.g., a valid user who has less than 4 books borrowed and no outstanding invoices will be selected from the database. Accordingly, a book is chosen that can be borrowed. Since the input data from the corresponding objects only contain to the identifier of an RFID tag, the tester needs to select the identifiers of the corresponding RFID tag from the test database and insert them in the template of the abstract test cases.

Since the formal description of the conditions is based on sets and relational algebra (see Section 4.2.4), an automatic selection of the test data is also conceivable. In order to achieve a technical implementation, the predicates would need to be converted into database-specific expression such as SQL. The result of the database query then provides a choice of valid input data. A representative should be selected and assigned to the test case. Although this approach seems to be clear, it was not further considered in this thesis, but is part of future work.

The model of the “rental process” use case illustrated in the previous section was transformed by means of a prototype implementation of ITERA the test case generation method and augmented with appropriate test data. The test case generation algorithm extracted several test cases. Each test case has been refined by the tester by providing the corresponding test data. The final test of test case 3 (see Figure 4.15) from the example library is presented in Listing 4.3. The test case consists of eight test steps (lines 12-19) that correspond to stereotyped input activities of the actor in the activity diagram. Lines 2-9 show the selected test data and additional inputs derived from the `ActorAction`’s class attributes. Lines 3-4 represent the tag identifiers of an user-id card and a book, which fulfil the criteria needed to execute the test case.

The automatic test execution framework uses virtual RFID tags, which are provided by the Rifidi toolkit (see Section 2.3.4). Beside the selected identifiers from the test database (parameter *id* in the listing), additional toolkit specific parameters are needed in order to create the virtual tags (*type* and *tagtype*). Analogous to these, virtual RFID readers are defined in lines 6-7. The Rifidi toolkit supports readers from different vendors with various configuration options. These are defined in a separate configuration file referenced through the *config* parameter.

To evaluate the test, application specific test drivers have been supplied for the GUI and the database (defined in Lines 8-9). In this example these are AutoIT, a framework for GUI automation and SQLite database connector.

Listing 4.3: TestCase 1 - Example Library "process rental"

```
 1 @SetUp
 2 TestData:
 3    userId1 = {type="RifidiTag", tagtype="EPCG1C2", id="3074257bf7194e4000001a85"}
 4    book1 = {type="RifidiTag", tagtype="EPCG1C2", id="6056257bf8194e4000011b99"}
 5 TestEnvironment:
 6    RFIDReaderConfig1 = {type="RifidiReader",name="UserIDReader",config="reader1.config"}
 7    RFIDReaderConfig2 = {type="RifidiReader",name="LibraryShelfReader",config="reader2.config"}
 8    AppplicationConfig1 = {type="AutoIT",name="LibraryGUI",config="libraryApp.config"}
 9    DatabaseConfig1 = {type="SQLLite",name="LibraryDBConnector",config="libraryDB.config"}
10
11 @Test diagram="ad_rentalprocess" name="tc1_rentalprocess"
12 TestAction GUIInputAction("rent books")
13 TestAction RFIDAddAction([userId1])
14 TestAction RFIDRemoveAction([userId1])
15 TestAction RFIDAddAction([book1])
16 TestAssertion DataBaseAssertion("[book1].status = rented AND [book1].rentals.user = [userId1]")
17 TestAction GUIInputAction("finish")
18 TestAction GUIInputAction("OK")
19 TestAction RFIDRemoveAction([book1])
```

The lines 12-19 are show the sequence of test steps which are applied to the SUT. According the the test path different inputs in form of `GUIInputAction` and `RFIDADDAction` (respectively `RFIDRemoveAction`) are listed. The concrete input parameters needed for executing the `execute()` method from the classes in the ITERA meta model (see Section 4.3.2) have been referenced accordingly.

4.6.3 Robustness Tests

Robustness tests presented here are not intended to exercise extreme cases, but rather to demonstrate fault-free operation of the application under modified environmental conditions. These are mainly due to the technical challenges involved in the operation of RFID systems, including unreliability of tag capture as identified in Chapter 3.

Most of the challenges stem from the physical effects that have to be addressed on the physical layer, however, some can also be handled by software, in particular through tag filtering and smoothing mechanisms. In order to provoke these effects and thus test correct filter and aggregation functionality, the derived tests cases need to be modified accordingly to reflect the modified environmental conditions and thus allow for an evaluation of robustness aspects. Currently, techniques used to eliminate unintended duplicate tag reads are often based on timing aspects and through changes of tags inside a interrogation zone, like first time appearances and disappearances.

Starting-point for the creation of RFID specific robustness tests are the test cases created in the previous section. These already represent valid use cases of the SUT and are therefore well suited to validate the correct implementation of the application with regard to the unreliability of the underlying RFID system. Goal of the robustness tests is to reduce the chance of unintentional multiple tag reads.

In the following, three classes of robustness test are defined by which the test cases are adapted. In principle, the modification of the test cases to reflect the input for an robustness test can be automated, but additional parameter boundaries of the individual modifications must be determined by the tester. These boundaries are described in the different robustness classes.

1. validity of filter and aggregation functionality
    a) temporal based event aggregation
    b) semantic based event aggregation
2. validity of bulk reading functionality

The first two classes (1a, 1b) examine the filter and aggregation functions used in the different components of the RFID system. A distinction is made between aggregations with temporal and aggregations with semantic relationships. In the case of time based aggregation, the reading events continuously generated by the presence of an RFID tag in the reading field of the RFID reader are aggregated over specified time frames (e.g. sliding window). With semantic based aggregations, the logical events are determined by the first appearance or disappearance of the RFID tag in the reading field. The third class (2) deals with the possibility of processing several RFID tags simultaneously, "as a bulk".

Temporal based event tag aggregation

The key characteristic of this test case modification is that the temporal dimension of individual activities on the test path is changed by the tester.

The aim of this class of robustness tests is to validate the filtering mechanisms used in the complete system, e.g. the RFID reader or middleware or application. Time-based filtering and aggregation techniques are often used to combine the continuous stream of RFID events into separate logical events and thus enable them to be assigned to distinct processes. The central question of this test is whether the filters and aggregation functions are able to reliably generate logical events from the raw data stream of an RFID reader even under changing conditions. Since all components and their interactions are already controlled by the test environment in the presented approach, this can be considered as a valid concept. The time course of the test, is determined by the individual test steps (in particular all subclasses of type `ActorAction`).

The test case's total duration is determined by the sum of individual intervals in the previously specified activities.

$$\sum_{a=a_I}^{a_F}(duration(a))$$

Where $a \in$ `ActorAction`, a_I is the start of node, a_F denotes the end node of the test path and a are all successor nodes traversed along the path.

The duration of a RFID tag inside the reading field can thus be determined by summing up all test steps between an (inclusive) RFIDAddAction and the corresponding (exclusive) RFIDRemoveAction.

$$\sum_{a=a_i}^{a_j}(duration(a))$$

Where a_i is of type `RFIDAddAction` and a_j of type `RFIDRemoveAction`.

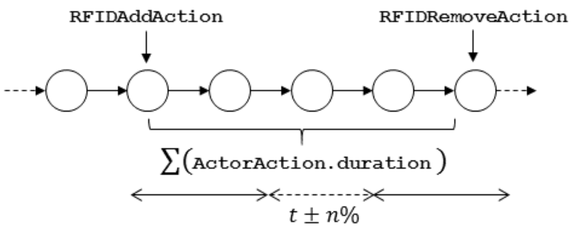


Figure 4.16: Determination and modification of the duration a tag is inside a reader's field

Figure 4.16 shows the procedure schematically. The duration between the two nodes and the proposed modification of timing aspects is depicted. To verify a reliable assignment of the RFID event to logical events, the retention time t is now changed according to a factor n defined by the tester. However, the necessary prerequisite for implementation is an exact specification to which extend the temporal boundary conditions may be violated. This can often be realized by means of a percentage value.

Semantic based event tag aggregation

The unintentional multiple reading of RFID tags without a temporal examination represents a further variant of robustness tests. In this case, the raw events generated by a RFID reader are not generated on the basis of temporal aspects, but by the first recognition of a RFID tag. Similar to the first occurrence of the RFID tag, the departure of a tag can also be used for distinguishing logical read events from the RFID data stream. Here, incorrect aggregation is caused by physical effects such as shadowing of RFID tags or interference in the radio field, often defined in the literature as "whitespots".

In contrast to the method for testing RFID applications presented in this thesis, manual testing with the aim to reproduce this effect is very difficult to perform. However, as the RFID environment can be completely controlled and manipulated by the test framework presented in Chapter 5, the system can be supplied with erroneous input that might result in multiple logical RFID events. To simulate these effects by means of a robustness test, a RFID tag is added and removed from the virtual reader several times. Figure 4.17 shows the procedure schematically. The central characteristic of this test class is that the underlying test case is complemented by additional test steps. The previously specified test path is manipulated so that between

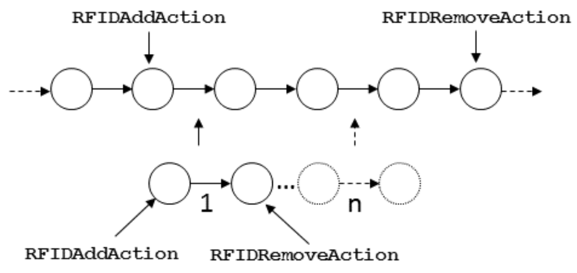


Figure 4.17: Robustness testing of semantic based aggregations

the original `RFIDAddAction` and `RFIDRemoveAction`, one or more repetitions of these activities are inserted. However, a prerequisite is that the tester specifies a maximum number of these repetitions.

Bulk reading

One of the fundamental advantages of RFID systems is the ability to capture multiple objects simultaneously. This is often referred to as bulk reading in literature. In contrast to other auto-identification technologies such as barcodes, this allows, for example, the scanning of entire pallets at dock doors. Subject of the third robustness test is therefore the examination whether the developed application copes with processing several RFID tags that are simultaneously in the reading field correctly.

The central characteristic of this modification of the test cases is that the number of input data of the RFID specific activities is changed in such a way that several RFID tags are in the reading field simultaneously.

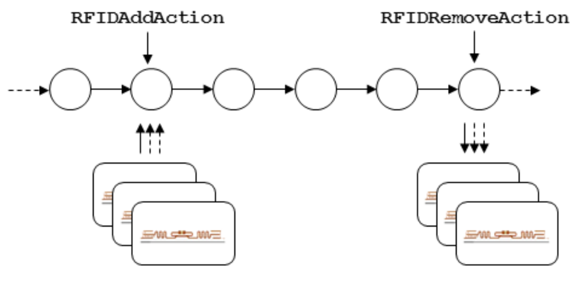


Figure 4.18: Robustness modifications for the validation of bulk reading

To check whether the application correctly implements this functionality, the `RFIDAddAction` (or `RFIDRemoveAction`) contained in the test path is executed with multiple RFID tags. However, testers must provide appropriate test data, i.e. valid RFID identifiers, in order to carry out the test. These can be either (semantically) valid or invalid entries, however, they still need to fulfil the input constraints (guard conditions) of the test obtained during test path generation. Figure 4.18 shows a schematic example test path generated through the test case generation algorithm. Here, the activities RFIDAddAction and RFIDRemoveAction are identified and the corresponding inputs are modified. These changes of the input data corresponds to the semantics introduced in Section 4.2:

$$\mathit{reader}(t_n) = \{id_1\} \cup \{id_2, \ldots\}$$

where $\{id_1\}$ is the original input data set and the set $\{id_2, \ldots\}$ the additional bulk tags of this robustness test.

In contrast to manual testing procedures, the ITERA method also allows very large tag populations to be inserted into the reader's interrogation zone. This enables not only the evaluation of bulk processing, but also provides the basis for performance tests. However, performance aspects have not been considered in the present thesis, but can be subject for further research.

4.7 Conclusion

A model based approach for testing functional requirements of RFID applications has been introduced. The key contributions of this thesis are comprising of a semantic model, a meta model to refine activity diagrams used for specifying tests, an algorithm to generate test cases and suggestions to select appropriate test data. Based on the foundation of a semantic model to describe the behaviour of the RFID system, a meta model which allows for refining activity diagrams to capture the intended functionality and bridging the gap between test model and RFID environment has been presented. The created models are transformed into directed graphs and traversed by a specially prepared algorithm that can cope with specific constructs of activity diagrams to create the test cases. Considering the constraints on the traversed execution paths appropriate test data is selected. In combination of the test cases and the test data executable test are derived which can be used as input for the test automation framework described in the next chapter.

In comparison to other MBT approaches 2.2.2, the unique aspect of the test approach examined here is the additional enrichment of the model with context parameters, which are a prerequisite for test execution. The test case generation examined in this thesis is based on the assumption that context factors, like timing, have to be taken into account. This applies in particular to testing, where test execution and the understanding of the impact of temporal behaviour is essential for the correct evaluation of RFID systems. The transformation of the resulting test models into test cases is a well known procedure, but with the special feature of integrating context parameters in the creation of RFID specific test cases and applying them during test execution. In addition, robustness tests are considered for the first time during the creation of the tests and are thus a novelty and a decisive contribution of this work.

5 ITERA Test Execution Framework

In the previous chapters the specific characteristics of RFID systems, in particular the unreliable detection, and the effects on development and operation have been explored. These effects also manifest themselves in an increased complexity of all quality assurance activities, from test modelling to test execution. So far, however, there are hardly any methods and tools that meet the special requirements for testing RFID applications.

A systematic way of creating test cases has been introduced. Systematically identified test cases are the fundamental prerequisite for high failure detection and effective testing. The aim is to test a large number of potential errors with the lowest possible number of test cases. From a theoretical point of view, test case identification is therefore the most important test activity.

In practice, however, test execution is usually much more important than the determination of test cases, since the test execution process has a decisive influence on the efficiency of the test. The total cost of the test is divided into two main areas: the cost of determining the test case and the cost of carrying out or evaluating the test. While the first part is "only" dependent on the complexity of the test object and the number of relevant test cases, the second part also grows almost linearly with the number of necessary tests and test repetitions. The number of these repetitions depends on the type of development process and not on the test methodology used. In iterative processes, such as those frequently occurring in practical RFID deployments, due to the gradual introduction of RFID systems into different areas of a business, the number of test repetitions required is naturally very high. In such cases, the testing effort for execution can only be minimized by automation. Furthermore, also the fact that some physical parameters cannot be influenced when testing in a real environment and that RFID tests have to be carried out in (near) real-time, in order to detect failures in the filtering of RFID data streams, emphasizes the great importance of test automation.

It was shown how abstract test cases can be generated from UML activity diagrams by model transformation and an algorithm to derive test paths using the ITERA methodology for testing RFID applications. Now it is to be shown that tests generated by this method can be supported by a test automation technology. The distinguishing feature of automated testing of RFID applications compared to traditional applications is, that - in addition to test data - a context environment (timing constraints from physical processes, unreliable tag capturing and the RFID readers) must be provided in the test environment during testing. The UML extension discussed in the previous section enables this data to be mapped in the test case model. Therefore, for their actual use in test execution, an automation tool must be able considering these parameters and provide the SUT with the artificial RFID events considering these context parameters.

In principle, different approaches can be used to simulate the context in which tests are carried out. One option for simulating these context parameters is based on providing a laboratory environment, that is capable of producing physical interferences (whitespots, erroneous read events) and is suitable for interacting with physical RFID readers. However, it can be assumed that this approach is generally uneconomical and technically challenging for some context parameters. Moreover, this approach would only be suitable for a limited number of testers, as such a laboratory environment is only available for a very small number of testers.

By extending the ITERA methodology with a virtual RFID environment, laboratories for the simulation are not needed for testing and thus the introduced approach becomes accessible for every tester of an RFID system. Therefore, the ITERA test methodology uses Rifidi to accomplish this task and allows for an automated test execution of the previously created test cases.

Parts of the subsequent sections have already been presented in [29].

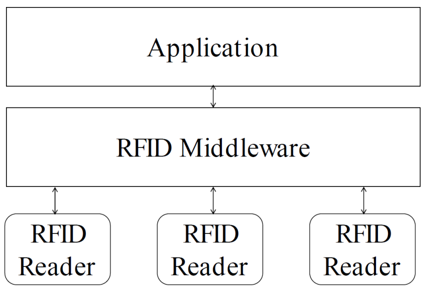
5.1 Virtual RFID System

To improve the test processes the real world RFID devices can be substituted by a virtual RFID environment, with virtual readers from the Rifidi Toolkit [128]. This allows an otherwise unchanged tool chain and creates a test environment as close as possible to the actual system environment. Being able to create such an environment is an essential prerequisite for system level testing (see Section 2.1.6). In Figure 5.1a the original system is shown, contrary to Figure 5.1b which shows the changed RFID system enhanced with virtual RFID readers. In the latter case, all RFID devices are substituted by virtual readers. This way a tester can now place virtual tags on the virtual readers, and generate the same input as with the real environment. But even further, this can also be used to reproduce the conditions while a test was carried out and to automate the test execution for the abstract test cases derived from the test model.

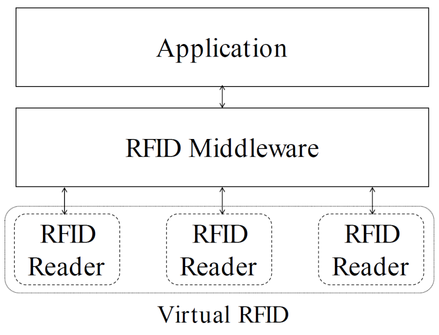
From the perspective of the RFID middleware and application the simulated system cannot be distinguished from a real system. This test environment can even be a system composed of randomly mixed real and virtual devices. However, then only the virtual RFID parts can be used for instrumentation through the extracted test cases.

5.2 Test Set-Up

As already shown in Section 3, in typical RFID systems the application logic is distributed between the RFID middleware and RFID application itself. Most of the logical parts are implemented in the application itself, but some filters, aggregations and evaluations are applied



(a) RFID System



(b) Virtual RFID System

Figure 5.1: RFID System vs. Virtual RFID System

on the reader's raw RFID events, before the events are processed by the application. RFID raw-events are reports the reader sends to the middleware. There are two modes, which can be distinguished, pull and push mode. In push mode it sends an raw-event every time a RFID tag leaves the RFID field, contrary to a RFID reader which is polled at a certain timing interval. The mode used depends on the configuration of the middleware.

In this approach the knowledge about how raw RFID events are created can be neglected, because the virtual RFID readers used are fully emulated RFID devices, which can be configured either way. Furthermore, the middleware is treated as part of the SUT, which allows to execute tests on system level. To achieve this, the *Test Harness* presented in Figure 5.2 is build around the middleware and application. A virtual RFID framework called *Rifidi Emulation*, which generates virtual RFID events as input for the middleware, is located inside of the *Test Harness*. It is instrumented by the instructions generated out of the test model. The evaluation of the test results is done through the *Evaluation Monitor*, which checks the RFID application's actions and results.

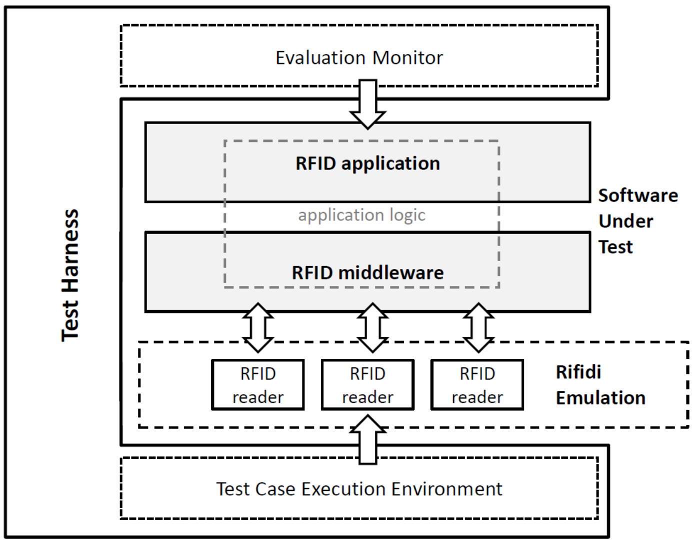


Figure 5.2: Schematic RFID Test Set-UP

The actual test execution is carried out by a so-called test engine, which serves as an instrumentation tool for the test procedure and takes over the execution of the test cases. It incorporates so called test drivers which connect to specific interfaces of RFID application.

5.3 ITERA Test Execution Framework

Tests above the component test level are performed by simulating the actual use of a software system or a component by the potential user. From a purely technical point of view, the process of carrying out the test is limited to manipulating the SUT by using tools in the same way as would be done by a human actor. Thus, the workflow has to be supported by a suitable simulation technology to provide corresponding inputs. Such technology is often called a test driver (see Patton [32]) in the terminology of software testing.

In the fourth step of the ITERA methodology the test cases that have been created from the activity diagram are executed. This is realized through application specific implementations of the classes from the ITERA meta model, so called test drivers, which are used in combination with a technical solution to stimulate and evaluate the SUT. As already pointed out, the test automation examined here does not make any concrete assumptions regarding the target technology and offers extension points, that allow other technologies to be used. The only requirement is that the test back-end technology to be used (such as Rifidi, AutoIt, SQlite) provides entry points in the meta model in form of subclasses of `<<TestAction>>`, explicitly `<<ActorInteraction>>` or `<<SystemAssertion>>`. If these requirements are met, they can be integrated into the automation tool during test execution.

In practice the test automation is of great importance, since the effort of testing is determined by the efficiency of the test execution to a large extent. In order to ensure automatic test execution, executable semantics have already been emphasized during the modelling of the test cases. All information necessary for the execution of test cases is already contained inside the test model in the form of stereotyped activities. Each `<<TestAction>>` represents one test step and contains test driver specific parameters to create the test environment, execute the different test steps or evaluate the results after their execution.

It is not a prerequisite that the test cases have been generated from a UML activity diagram. Therefore, manually created abstract tests (test scripts) can also be used as valid input for the test execution framework.

During the transformation of the activity diagram to test cases, the procedure discussed in Section 4.5.3, a set of execution paths were generated for each activity diagram. These paths consist of sequences of *TestAction* elements, each containing an executable method that allows the test execution module to execute the test step. The source code listing 5.1, shown below, shows the abstract test case generated during the example library “rental process” used throughout the introduction of the ITERA Method in Chapter 4.

Listing 5.1: Abstract TestCase 1 - Example Library “rental process”

```
@SetUp
TestData:
    userId1 = {type="RifidiTag", tagtype="EPCG1C2" id="3074257bf7194e4000001a85"}
    book1 = {type="RifidiTag", tagtype="EPCG1C2" id="6056257bf8194e4000011b99"}
TestEnvironment:
    RFIDReaderConfig1 = {type="RifidiReader",name="UserIDReader",config="reader1.config"}
    RFIDReaderConfig2 = {type="RifidiReader",name="LibraryShelfReader",config="reader2.config"}
    AppplicationConfig1 = {type="AutoIT",name="LibraryGUI",config="libraryApp.config"}
    DatabaseConfig1 = {type="SQLLite",name="LibraryDBConnector",config="libraryDB.config"}

@Test diagram="ad_rentalprocess" name="tc1_rentalprocess"
    TestAction GUIInputAction("rent books")
    TestAction RFIDAddAction([userId1])
    TestAction RFIDRemoveAction([userId1])
    TestAction RFIDAddAction([book1])
    TestAssertion DataBaseAssertion("[book1].status = rented AND [book1].rentals.user = [userId1]")
    TestAction GUIInputAction("finish")
    TestAction GUIInputAction("OK")
    TestAction RFIDRemoveAction([book1])
```

The ITERA test execution framework is divided into three components, namely the TestPreprocessor, the TestEngine and several TestDrivers. The TestPreprocessor and TestEngine are essential parts of the ITERA test methodology and thus must not be customized. In contrast to the TestDrivers, which might be very application specific as they are build on a concrete technology like Rifidi or a SQL Library.

ITERA Test Preprocessor - The *TestPreprocessor* does not execute the test cases them selves, but translates the abstract test case description into an executable form for the test engine. For each stereotype defined in the ITERA meta model it creates an appropriate instance of an object, which can then be executed by the ITERA test engine. In the prototype implementation this functionality is realized by using reflections[1].The generated test cases (see: source code listing 5.1) are provided as input and parsed to create instances of the classes for each stereotyped test step accordingly.

ITERA Test Engine - The *TestEngine* is the actual execution part of the test automation framework of ITERA. It uses the results of the *TestPreprocessor* and executes each test step by calling either the method `ActorInteraction.execute()` or `SystemAssertion.assert()`. Each test step is implemented as a *TestDriver* in an application specific test automation technology, like “Rifidi” or “SQL”. This way the *TestEngine* is kept simple and does not need to know application specific details. Furthermore, it is independent from the actual tests and can be used for various test scenarios.

Test Driver - The TestDriver is a concrete implementation of of either a `ActorInteraction` or `SystemAssertion`. Each class provides an entry point, where functionality of the test automation technology can be realized. The ITERA methodology does not restrict the test execution to predefined interfaces, therefore, several *TestDrivers* can be used. Each is executing a certain

[1]Reflections are the ability of an programming language, like Java, to inspect classes during runtime, which can be used to dynamically create instances of these. This way not all details of the program need to be available during compile time.

test action and utilizes application specific simulation toolkits, e.g. “AutoIT” or “Rifidi”. The evaluation of the test is handled similar by including specific interfaces to the SUT, like “SQL”. Some concrete implementation examples of these TestDriver’s are shown in Section 5.4.

5.4 Test Drivers and integration of the virtual RFID environment

The Rifidi Emulator was integrated into the ITERA test execution framework in order to support the RFID specific execution of test cases derived using the ITERA method. The virtual RFID readers emulated by the Rifidi Environment allow to create a fully controllable RFID environment, which is not only capable to stimulate the SUT with artificial RFID events as needed for testing RFID applications, but is also suitable for system testing as the readers can hardly be distinguished from real readers. Therefore, the Rifidi Toolkit was chosen to create the virtual RFID system as introduced in Section 5.1. The core functionality is provided by the *Emulator Engine*, which enables the creation of virtual RFID readers, RFID tags and thus also artificial RFID events. However, in order to perform RFID application tests using this virtual RFID system, the newly created virtual environment needs to be instrumented by the test cases generated previously.

Rifidi is capable to emulate different reader protocols (like AlienProtocol or LLRP) and the functional behaviours of physical RFID readers. The subclasses of stereotype `RFIDAction` are used to model the interactions with RFID readers. Each `RFIDAction` can also be represented as a tuple $RFIDEvent := (RFIDReader, Time, RFIDTags)$. Whenever a `RFIDAction` is executed during testing, Rifidi is instrumented to emualte the raw RFID events. The raw-events generated depend upon the current configuration of the RFID reader. In this approach the reader is configured by the RFID middleware. This conception is valid, since the RFID middleware is part of the SUT.

The abstract test cases derived from the test model can be used to interact with the RFID environment, as they already contain sequences of activities with explicit semantics, in particular where RFID tags are placed inside a reader’s interrogation zone. In order to achieve this, the virtual environment is integrated into the ITERA test methodology and realized in the form of a specific Rifidi test driver. This test driver is an implementation of `<<ActorInteraction>>` and provides the functionality of the `<<RFIDADDAction>>` and `<<RFIDRemoveAction>>` (in the meta model in Section 4.3.1 as a generalisation of `<<RFIDAction>>`).

As a general example, the implementation of the RFID-specific sterotypes using the Rifidi Toolkit as RFID specific back-end is shown in 5.2 and 5.3:

Listing 5.2: Example source code of the TestDriver RifidiAddAction

```
public class RFIDAddAction extends RFIDAction
```

```
{
   RifidiConfig config;
   RifidiReader reader;

   public RFIDAddAction(TestConfig config)
   {
      this.super(config);
      this.config = (RifidiTestConfig) config;
      reader = ReaderManager.getInstance().getReader(config.readerName);
   }

   public void execute(RifidiTestData input){
      RifidiTag rfidTag = input.getRifidiTag();
      reader.add(rfidTag);
   }
}
```

Listing 5.3: Example source code of the TestDriver RifidiRemoveAction

```
public class RFIDRemoveAction extends RFIDAction
{
   RifidiConfig config;
   RifidiReader reader;

   public RFIDAddAction(TestConfig config)
   {
      this.super(config);
      this.config = (RifidiTestConfig) config;
      reader = ReaderManager.getInstance().getReader(config.readerName);
   }

   public void execute(RifidiTestData input){}
      RifidiTag rfidTag = input.getRifidiTag();
      reader.remove(rfidTag);
   }
}
```

Beside the advantages of being able to simulate the RFID environment, something which is barely realizable in reality, the extension of the methodology by using Rifidi also allows a detailed logging of test execution. All communication between the RFID system (application and middleware) and the RFID readers can be logged and analysed. In comparison to manual testing this can be a huge advantage. Furthermore, it also allows for debugging of the SUT, as all information and the exact communication between the components, usually hidden as most RFID readers are embedded devices, is available for inspection.

It is challenging to evaluate the application due to the diverse variety of RFID application areas. Compared to RFID middleware systems, where the ALE [85] standard can be used to communicate and evaluate the test results, for each application a distinct evaluation monitor adapter needs to be implemented. In addition to the missing interface for the evaluation of the test results in applications, even more complexity is introduced by the nature of RFID systems (see Chapter 3).

The evaluation function of the appropriate test steps `SystemAssertion.assert()` can be realized by a simple query on the database. In the library example, the predicate defined through test case refinement and test input data determination to ensure that a book is rentable, in the set-based semantics defined as *book* ∈ *rentable*, is translated into an SQL statement like:

```
SELECT status FROM library WHERE BookID=\3074257bf7194e4000001a85";
```

In this case, since the book is supposed to be in the table, the result set needs to be identical to *rentable*. So, the different process models are analysed and the corresponding test steps of stereotype `SystemAssertion` are used to generate the required test code for the test case. After the execution of the test case the reports of the *TestEngine* can be checked to validate the correct functionality of the SUT.

An example of a concrete implementation of a `SystemAssertion` as subclass `DatabaseAssetion` is shown in listing 5.4:

Listing 5.4: Exmaple source code of the TestDriver DataBaseAssertion

```java
public class DataBaseAssertion extends SystemAssertion
{
    DatabaseConfig config;
    String predicate;
    Connection conn = null;

    public DatabaseAssertion(DataBaseConfig config, String predicate)
    {
        this.config = (DataBaseConfig) config;
        this.predicate = predicate;
        try{
            String url = config.getDBURL();
            conn = DriverManager.getConnection(url);
        }catch(SQLException e)
            e.printStackTrace();
        }

    }

    public boolean assert()
    {
        String sql_predicate = Helper.parsePredicate(predicate);
        try{
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql_predicate);
            if(rs.next()){
                return true;
            }else
            {
                return false;
            }
        }catch(SQLException e){
            e.printStackTrace();
        }finally
        {
            if ( conn != null )
                try { conn.close(); } catch ( SQLException e ) { e.printStackTrace(); }
        }

    }
}
```

5.5 Conclusion

To conclude the ITERA test framework, most importantly a virtual RFID environment has been incorporated to allow for the generation of RFID specific stimuli for the SUT. The evaluation of the test outcome has been realized through the implementation of appropriate interfaces, like SQLLite. Similar test drivers are used to instrument and evaluate the graphical interface.

The test execution framework can be easily extended by including corresponding technologies and concrete implementations of the `ActorInteraction` or `SystemAssertion` of the ITERA meta model. This enables the tester to extend the use of the ITERA test methodology to application areas with additional sensors or actors, which have not been considered during the development of the approach in this thesis.

The executable semantics of the test model, i.e. the refined activity diagram with the stereotyped activities, provide an straight forward method to execute the tests. The TestPreprocessor and the TestEngine can also be used by manually created tests. If appropriate technologies have been supplied, the test execution and evaluation can be performed automatically. This way efforts for test execution can be decreased and more time can be spend on the design and specification of the tests.

6 Evaluation of the ITERA Test Methodology

The previous chapters introduced the ITERA test methodology which is used to create activity diagrams, refine the semantics of the activities and include test data, generate test cases and execute tests derived this way in an automatic fashion, mainly by using an virtual RFID environment.

The evaluation of the ITERA method with regard to the research questions, its feasibility and practical benefits is now the subject of this chapter. In the beginning the evaluation is concerned with assessing the methodology in regard to the research questions raised in Section 1.3. Afterwards a feasibility study is carried out to demonstrate the applicability of the method. In addition, the algorithm (see Section 4.5.3) used to generate test cases will be validated in terms of the ability to produce valid test cases which fulfil the defined coverage criteria. Finally, the extent to which the ITERA methodology meets RFID specific requirements and an inventory of past experience will be carried out to determine whether something can be done better in a different way.

6.1 Evaluation of the ITERA Methodology in regard to the Research Questions

An evaluation of ITERA method should be based on the research question raised at the beginning (see Section 1.3): "How can RFID Applications be tested?". With regard to the approach presented in this thesis, there is a clear answer: A systematic creation of RFID-specific tests was discussed in detail in Chapter 4 and the subsequent execution of these tests in Chapter 5.

The research questions subordinated to this question can also be answered clearly.

- How can RFID applications and their input be described in regard to testing?
- Is there a systematic way of choosing test cases?
- How to simulate the interaction between the actors of an RFID System and the Software Under Test to allow automated test execution?

RFID specific challenges were identified in Chapter 3, a semantic model and a systematic way to derive test cases was introduced in Chapter 4 and a suitable test automation, desperately needed for testing, especially in regard to reproducible test and cost efficiency was presented in Chapter 5.

6.1.1 How can RFID applications and their input be described in regard to testing?

The challenges of RFID systems considering requirements for testing RFID applications, as well as, methodical and technical solutions for test case generation and automation were considered during the investigation of the RFID system analysis (see Chapter 3). This way, a test method was developed that not only meets the needs of RFID system level testing, but also addresses the usual set-ups in terms of RFID hardware and software solutions used in practice.

In addition to address related work, a distinction was made between conceptual and technical problems of testing RFID applications:

The conceptional challenges include the absence of appropriate test methodologies suitable for testing RFID applications and the typical infrastructure of RFID systems. Due to the complex and distributed architecture of RFID systems, system level testing can be seen as an important aspect to improve the quality of the RFID application. Especially as there are many factors, like configuration settings of the individual readers and the middleware, which influence the correct interaction of the sub-systems and can only be tested when all components are finally composed to a complete solution. In the proposed methodology, these have been taken into account by providing a suitable test harness surrounding the complete system and through the systematic generation of test cases based on the specification. As needed for system level testing the derivation of tests is based on functional requirements which can be modelled sorely on the basis of the requirements specification and without the a insight on the concrete implementation. The test harness provided by the virtual RFID environment and extendable test execution framework in order to execute the tests in a black box fashion, including simulated interactions between the user and the SUT has been presented.

Technical challenges include factors that make it difficult or impossible to methodically test RFID applications. These include physical issues each RFID system faces, as the unreliability RFID event capturing. In particular the provision of an reproducible execution context which includes factors that cannot be controlled with reasonable effort, like false positive and false negative tags reads. By using the ITERA methodology, the challenges are addressed through the fully controllable RFID environment embedded in the test execution framework and by providing a systematic way to modify tests to reflect robustness aspects that provoke unintentional tag readings. This way the RFID application can also be evaluated in regard whether these technical challenges can be coped with properly.

Although a large number of papers already consider model-based testing of applications (see Section 2.2.2), a comparatively comprehensive solution that considers modelling in the concrete context of RFID systems, has not yet been the subject of research. Unlike currently available RFID related literature with focus on testing (see Section 2.3.5), the approach presented in this thesis enables system level testing of RFID applications, which has not been addressed yet. These publications mainly focus on testing RFID middleware, performance evaluation, or simulation of supply chains. In contrast, the approach developed during this thesis, provides systematic derivation of test cases and addresses test specific challenges of RFID application testing, including test-relevant context parameters and oracles for the evaluation of the SUT.

6.1.2 Is there a systematic way of choosing test cases?

Beside the semantic set-based model for RFID systems, a UML-based meta model for the specification of RFID tests has been presented. The use of this meta model does not only allow for modelling tests and provide semantics to the test models, but also takes parameters into account with relevance to testing during test specification in a common document. The presented meta model is an essential part of modelling RFID specific tests and integrates smoothly into the development process through the use of notations like UML, which is widely adopted in practice. In addition to the integration of test data in to the model, a executable semantics of the activity diagrams, which are used to describe the use cases, has been emphasised. Thus this does not only enable to execute the tests automatically, but also provides extension points to integrate other test technologies which have not been subject in this thesis.

A method for generating test cases from UML activity diagrams has been presented. It is based on the transformation of UML activity diagrams to directed graphs, that are used to extract possible execution paths through the activity diagram. During the test generation it is ensured that the generated test cases cover the defined coverage criteria, i. e. each modelled activity is executed by at least one test case, and thus systematic tests are derived. Furthermore, it is also ensured that every cycle contained in the test model is run at least once and actions on concurrent control flows are interleaved in such a way, that they represent a permissible execution sequence from a perspective of a human user interacting with the RFID application.

6.1.3 How to simulate the interaction between the actors of an RFID System and the Software Under Test to allow automated test execution?

An RFID toolkit, which lowers the efforts during test execution, allows for cost savings in both development and testing and enables testing of RFID applications was integrated into the ITERA methodology. The Rifidi Emulator not only provides virtual RFID devices from different vendors and but also supports common standards used in today's RFID middleware solutions. The combination of the proposed test methodology and the virtual RFID environment not only supports testing of RFID applications in various ways, but also enables test automation.

Especially for RFID, the need for automatic test execution is high, because often the tests have to be carried out in real-time. Due to the unpredictable nature of the radio field, the integration of the virtual RFID environment allows for reproducible conditions during testing. Which can otherwise only be achieved through tremendous efforts or with a controllable environment, like a dedicated laboratory equipment. Moreover, the approach presented enables to test RFID applications, that could previously only be tested manually, supports agile development and eases the testing by supplying a instantly available test environment without conflicting with the development of the application.

6.1.4 Summary and Results

A novel methodology for the system level test of RFID applications has been proposed. The approach called ITERA, allows for the automatic generation of tests, defines a semantic model of the RFID system and provides a test environment for RFID applications. The method introduced can be used to gradually transform use cases into a semi-formal test specification. Test cases are then systematically generated, in order to automatically execute them in the test environment.

RFID specific requirements for testing have been analysed and identified. The system level test approach uses the principle of separation of concerns and black-box testing in combination with a virtual environment for test execution. The models used for the test cases are specified with UML, which allows for an exchange between tester, requirements engineer and stakeholders. The physical nature of processes in reality which build the basis of RFID event generation have been considered. The presence of RFID tags in an area, monitored by an RFID reader, can be modelled by time-based sets using set-theory and discrete events. The test execution can be realized through test drivers, which implement concrete interfaces to the application, e.g. a simple SQL query. Additional application specific needs can be integrated through the executable semantics of the meta model and the resulting test model. Furthermore, the proposed model based test method and the semantics of the appropriate test steps can be used to specify RFID system and their applications in regard to testing providing a comprehensive procedure, starting with the test specification up to the automated test execution.

The approach introduced in this thesis allows to reduce the efforts for RFID application testing by systematically generating test cases and the automatic test execution. In combination with the developed meta model and by considering additional parameters, like unreliability factors, it not only satisfies functional testing aspects, but also increases the confidence in the robustness of the tested application. Mixed with the instantly available virtual readers, it has the potential to speed up the development process and decrease the costs - even during the early development phases. ITERA can be used for highly automated testing, reproducible tests and because of the instantly available readers, even before the real environment is deployed. Furthermore, the total control of the RFID environment enables to test applications which might be difficult to test manually.

6.2 Applicability and feasibility of the ITERA Method

In the following sections the presented approach will be examined with regard to the applicability of the method on the basis of two feasibility studies as well as the correctness of the test case creation.

The applicability of the presented approach to modelling RFID specific test models was already addressed during the description of the methodology in Chapter 4. This was done implicitly on the basis of the continuously used example “rental process” in the RFID-equipped library. To demonstrate that the approach presented can also be applied in other applications, two case studies are presented in Section 6.2. The selection of applications was based on the classification of RFID applications presented in Section 3.6. The aim is to evaluate the model creation on the basis of the description of further use cases.

It is not a claim of the proposed method that it can be used to determine test cases which guarantee a higher level of error detection. The quality criterion considered is the suitability of the test cases and test suites for testing the specified individual requirements. In this context, “suitability” is to be understood in particular in such a way that the number of test cases is suitable for demonstrating the testing of individual requirements to stakeholders, and in particular that the proposed procedure is advantageous compared to other test methods. The goal of the presented method is the systematic and structured derivation of test cases. As with all other constructively applicable test methods and model-based approaches to the generation of test cases, it is not possible to make a direct statement as to whether parts of the system to be tested that are particularly faulty are given priority in the test. The specified use cases are representative examples of the requirements for an RFID application defined by the specification. Considering this, systematic test cases fulfilling a coverage metric can be an indicator for “suitability”. Therefore, it is to be shown that the algorithm proposed generates test cases, which fulfil the defined coverage criteria.

In the following sections, two application examples will be examined to show the applicability of the ITERA method. For this the continuously used example “Library” and a “Retail Store” serve as the basis for the case study. The examples examined during the evaluation correspond to the categories identified in Section 3.6:

- Historically oriented tracking applications (retail shop example)
- Adaptive real-time or control flow & status-based applications (retail shop example)
- Interaction-based applications (library example)

The feasibility studies investigate the practical application of ITERA on real-world oriented scenarios. Its aim is to show that, real-world application can be modelled using the ITERA. For this, a challenging subset of the use cases from “Library” and “Retail Store” are used.

6.2.1 Feasibility Study: "Retail Store"

In this section a case study, in a retail clothing shop, is shown to evaluate the proposed methodology on testing RFID applications. First the example application is explained, then the different models are created and finally the test code is generated from the extracted models. The work presented here is partly published in [120].

Use Case Description

The store, presented in Figure 6.1, consists of two rooms, a warehouse and a salesroom. The shop sells t-shirts and sweaters. The products for sale are delivered directly to the warehouse. A delivery truck arrives and unloads the goods to the warehouse, where they are placed until they are available for selling in the salesroom. During the delivery, all goods are moved through *RFID Gate 1*, located at the warehouse's entrance. To refill the stock of the salesroom, products are picked up from the warehouse and moved into the salesroom. *RFID Gate 2*, located at the door of the salesroom, monitors the goods transported into the salesroom. Eventually, a customer wants to buy a product, the chosen product is moved to the sales counter (*RFID Reader 3*) and sold to the customer. The customer leaves the store, with the newly vested goods, through the exit/entrance of the salesroom, where an anti-theft system, consisting of *RFID Gate 4*, is located.

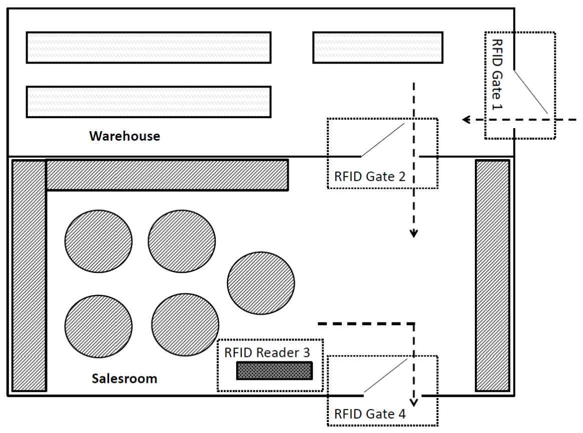


Figure 6.1: Use Case: Retail Store Layout

Data Model

The RFID application itself, is based on a database. A snapshot is presented in Table 6.1. Each row represents exactly one stock item in the store. There are two product-types: sweaters and t-shirts. Each product contains an ID, which is equivalent to the RFID Tag's identifier, attached to the physical item.

As it can be seen in Table 6.1, each product has an associated state:

TagID	ProductType	State
0001	sweater	warehouse
0002	sweater	warehouse
0003	sweater	salesroom
0004	t-shirt	salesroom
0005	t-shirt	sold

Table 6.1: Demo Application's Database Snapshot

- **warehouse**: the product is currently in the warehouse
- **salesroom**: the product is currently in the salesroom
- **sold**: the product was sold (kept in DB for accounting purposes)

The abstraction function id defined as shown in Section 4. E.g. The *State* of the product with the *TagID* = "0001" is warehouse $State(0001) = warehouse$ and `ProductType` is sweater $ProductType(0001) = sweater$.

In this approach UML class diagrams are used to formally represent the business domain. The Domain Model contains two types of products: `T-Shirt` and `Sweater`. Each product has a `RFIDTag` attached to it, which contains a unique identifier. The stock of goods of each room is represented in the class model by the `Warehouse`, respectively the `Salesroom`. The `Sales` class is used to collect all sold items for accounting purposes. After a accounting year, the sold products will be removed from the system's database. The stereotype *RFIDEnabled* is used to detect RFID Tag information and allows automatic generation of test code.

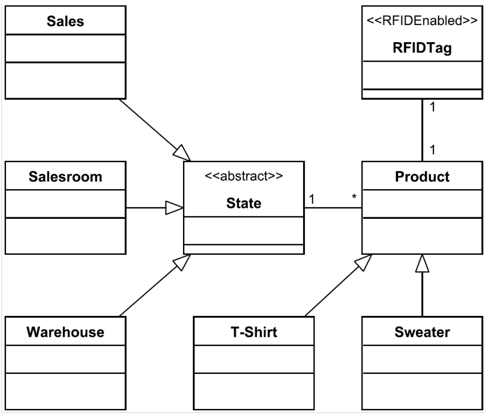


Figure 6.2: UML-class model of the retail shop application

Business Processes

The application is based on four business processes:

1. **delivery**: Once goods are delivered to the warehouse and pass RFID Gate 1, the products detected need to be added to the warehouse.

2. **refill_salesroom**: Once goods are monitored at the RFID Gate 2, the RFID application is supposed to remove them from the warehouse's stock and add them to the salesroom's stock to represent the current location of the goods and give an accurate view on the stocks.
3. **sell**: Once products are sold at the sales counter (RFID Reader 3), they need to be marked sold (the product's state changes to sold) by the application.
4. **anti_theft**: When goods are detected at RFID Gate 4, the application is supposed to check if the product is sold. If not the alarm of the exit gate needs to be activated.

Process Models

As demonstrated in Section 4.1 the processes of an use case can be used to derive systematic test cases. A Process Model, which is a UML activity diagramm, describes a combination of user and system interactions. The user interaction generates the input and the system interactions describe the changes of the SUT. The following Process Models are modelled accordingly to the business processes presented in Section 6.2.1. The basic principle is to refine the processes and make the in- and outputs explicit. Afterwards, the activities are refined using the stereotypes using the ITERA meta model.

Process delivery - The application's input in process *delivery*, presented in Figure 6.3, is described by a stereotyped action `<<RFIDAddAction>>`, which is created by the activity "move products to warehouse". The new item which will be put in the warehouse is denoted by p, which is of type `Product`. In this example each product has a RFID tag attached to it. Therefore, the class is an valid input for the subclasses of `RFIDAction` and allows to select the identifier for a virtual RFID tag and enables test code generation. The RFID reader *RFIDGate1* is associated with the business process *delivery* and is used to refine the `<<RFIDAddAction>>`. The resulting change of the RFID application is described by the activity "update warehouse stock" stereotyped `<<DatabaseAssertion>>`. In the semantics defined in 4.2.4 this is equivalent to $W(t_{i+1}) := W(t_i) \cup \{p\}$. All other sets, if not explicitly listed, remain unchanged (e.g. $S(t_{i+1}) := S(t_i)$).

By applying the ITERA test case generation algorithm on the activity diagram (shown in Figure 6.3) the following test cases are extracted:
TC1: I $\rightarrow$ A1 $\rightarrow$ A2 $\rightarrow$ S1 $\rightarrow$ D $\rightarrow$ S3 $\rightarrow$ M $\rightarrow$ A4 $\rightarrow$ F
TC2: I $\rightarrow$ A1 $\rightarrow$ A2 $\rightarrow$ S1 $\rightarrow$ D $\rightarrow$ S2 $\rightarrow$ A3 $\rightarrow$ M $\rightarrow$ A4 $\rightarrow$ F

Depending on the conditions traversed ([product ordered] and [product unknown]), two different RFID tags have to be selected by the tester. One tag which is known by the SUT and represents a product that has been ordered and one identifier which corresponds to a product that has not been ordered.

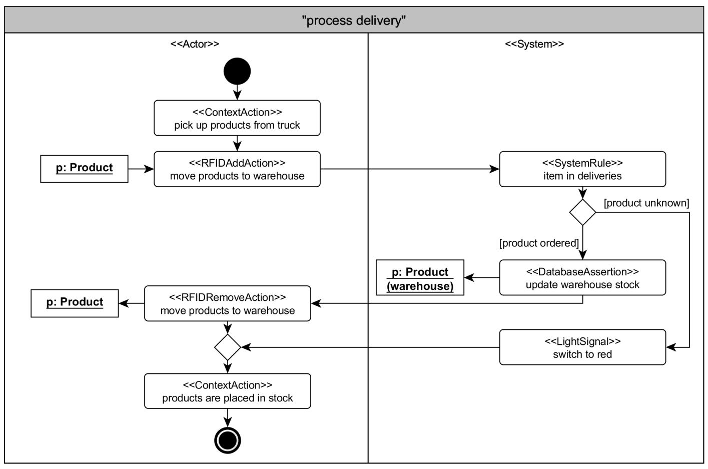

"process delivery"
<<Actor>>
<<System>>
<<ContextAction>> pick up products from truck
p: Product
<<RFIDAddAction>> move products to warehouse
<<SystemRule>> item in deliveries
[product unknown]
[product ordered]
p: Product (warehouse)
<<DatabaseAssertion>> update warehouse stock
p: Product
<<RFIDRemoveAction>> move products to warehouse
<<LightSignal>> switch to red
<<ContextAction>> products are placed in stock

Figure 6.3: Process Model "delivery"

Process refill_salesroom - The application's input of process *refill_salesroom*, shown in Figure 6.4, is described by a stereotyped action `<<RFIDAddAction>>`, which is shown as "products are moved to salesroom". Before the generation of the test cases appropriate identifiers of product, in this case p of type `Product`, need to be selected to extract the TagID which serves as input for the test execution framework. Assuming that the actions of the previous business process has been successful completed, it can be assumed $p \in W(t_i)$. The occurrence of the Product with it's attached RFID tag triggers the execution of the corresponding activity "update database stock" of type `<<DatabaseAssertion>>`. This is semantically equal to: $W(t_{i+1}) := W(t_i) \setminus \{p\}$, $S(t_{i+1}) := S(t_i) \cup \{p\}$

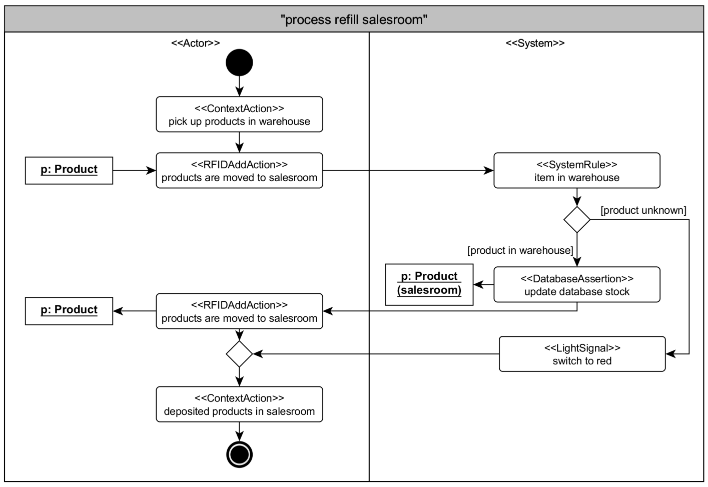

"process refill salesroom"
<<Actor>>
<<System>>
<<ContextAction>> pick up products in warehouse
p: Product
<<RFIDAddAction>> products are moved to salesroom
<<SystemRule>> item in warehouse
[product unknown]
[product in warehouse]
p: Product (salesroom)
<<DatabaseAssertion>> update database stock
p: Product
<<RFIDAddAction>> products are moved to salesroom
<<LightSignal>> switch to red
<<ContextAction>> deposited products in salesroom

Figure 6.4: Process Model "refill_salesroom"

Similar to the first business process the ITERA algorithm for test case generation is applied on the activity diagram shown in Figure 6.4. TC1 : I → A1 → A2 → S1 → D → S3 → M → A4 → F

TC2 : I → A1 → A2 → S1 → D → S2 → A3 → M → A4 → F

Here again, test input data fulfilling the conditions [product unknown] and [product in warehouse] is selected by tester to complete the abstract test cases which allow to execute the tests in the ITERA test execution framework.

Process sell - In this process the customer wants to buy some products from the shop. After some goods have been selected, he proceeds to the sales counter. The application's input generated by the process *sell*, shown in Figure 6.5, is represented by the activity "place items on the sales counter", therefore the stereotype `<<RFIDAddAction>>` was applied. From a process point of view, the customer places the products on the sales counter and pays for them. The changes within the SUT are reflected by the activities "calculate price", "display amount", "amount correct" and "update salesroom stock". The activity "calculate price" is not really test relevant as the focus is only on testing RFID specific details of the application under test, therefore no stereotype from the meta model was applied to it. However, other activities have influence the test. The system needs to assure that the correct amount was paid. Therefore, the activity "amount correct" is of type `<<SystemRule>>`. If the transaction was successful the database of the shop is modified and the appropriate products are marked "sold". The activity "update salesroom stock" of type `<<DatabaseAssertion>>` reflects the changes in the database. In the set based semantics of this can be described as: $S(t_{i+1}) := S(t_i) \setminus \{p\}$, $O(t_{i+1}) := O(t_i) \cup \{p\}$. If the amount was not correct the process is either repeated or cancelled.

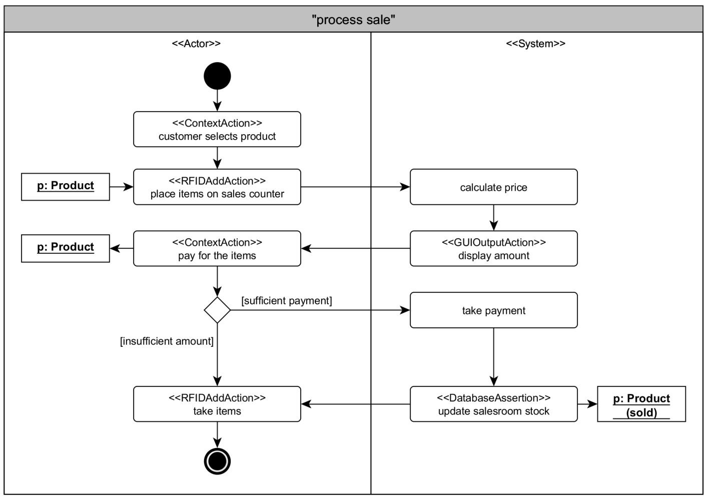

"process sale"
<<Actor>>
<<System>>
<<ContextAction>> customer selects product
p: Product
<<RFIDAddAction>> place items on sales counter
calculate price
p: Product
<<ContextAction>> pay for the items
<<GUIOutputAction>> display amount
[sufficient payment]
take payment
[insufficient amount]
<<RFIDAddAction>> take items
<<DatabaseAssertion>> update salesroom stock
p: Product (sold)

Figure 6.5: Process Model "sell"

The algorithm generated two test cases, as expected:
TC1: I $\rightarrow$ A1 $\rightarrow$ A2 $\rightarrow$ S1 $\rightarrow$ S2 $\rightarrow$ A3 $\rightarrow$ A4 $\rightarrow$ F
TC2: I $\rightarrow$ A1 $\rightarrow$ A2 $\rightarrow$ S1 $\rightarrow$ S2 $\rightarrow$ A3 $\rightarrow$ S3 $\rightarrow$ S4 $\rightarrow$ A4 $\rightarrow$ F

In test case 1 the calculated amount could not be paid by the customer. Therefore the process was aborted. Test data is selected by the tester according to this guard condition. Test case 2, on the other hand, corresponds to the successful scenario in which the products selected by the customer were sold. Appropriate test input is supplied by the tester.

Process anti_theft - The RFID input generated by the process *anti_theft*, shown in Figure 6.6, this is reflected by the activity "customer leaves store" stereotyped with `<<RFIDAddAction>>`. Depending on the successful completion of the previous business process, the product p is in the set salesroom or in sold. Equally to $p \in S(t_i)$ or $p \in O(t_i)$. The actions during this business process can be modelled as: $\{p\} \in S(t_i) \Longrightarrow S(t_{i+1}) := S(t_i) \backslash \{p\}$. The implication allows to distinguish the actions the SUT is supposed to execute depending on the set p is in. The evaluation of additional actors, like the activation of the alarm, is part of future work and not considered in this approach. In order to achieve this, an extension of the class `<<SystemAssertion>>` would allow to extend the functionality.

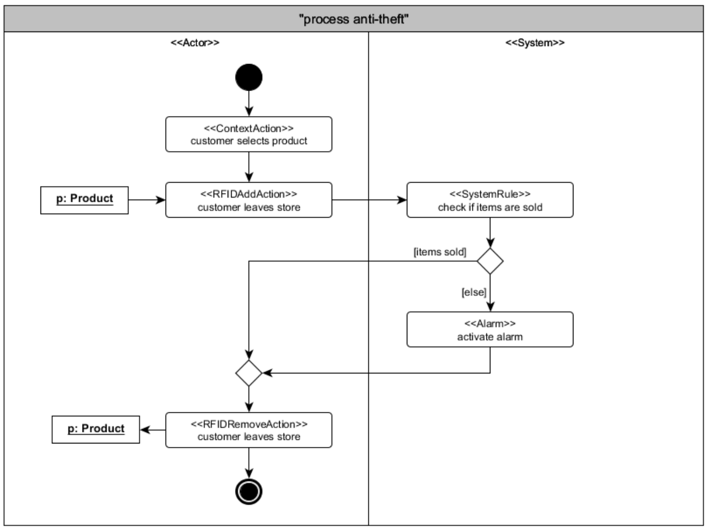

"process anti-theft"
<<Actor>>
<<System>>
<<ContextAction>> customer selects product
p: Product
<<RFIDAddAction>> customer leaves store
<<SystemRule>> check if items are sold
[items sold]
[else]
<<Alarm>> activate alarm
p: Product
<<RFIDRemoveAction>> customer leaves store

Figure 6.6: Process Model "anti_theft"

The extracted test paths are listed below.
TC1: I $\rightarrow$ A1 $\rightarrow$ A2 $\rightarrow$ S1 $\rightarrow$ D $\rightarrow$ C [items sold] $\rightarrow$ M $\rightarrow$ A3 $\rightarrow$ F
TC2: I $\rightarrow$ A1 $\rightarrow$ A2 $\rightarrow$ S1 $\rightarrow$ D $\rightarrow$ C [else] $\rightarrow$ S2 $\rightarrow$ M $\rightarrow$ A3 $\rightarrow$ F

S2 in test case 2 represents the functionality of the alarm. To evaluate the activation of the alarm, a product that has not been sold has to be selected by the tester in test case 2.

6.2.2 Feasibility Study: "Library"

In this section a case study, in a RFID equipped library, is performed to demonstrate the applicability of the proposed methodology on testing RFID applications. First, the general set-up of the library example use case is explained, then an already refined activity diagram representing an use case of the RFID application with increased complexity is presented.

Starting point of this case study is a library equipped with RFID readers and RFID tags. Beside the RFID tags used in books, user ID cards with embedded RFID tags were also issued. To decrease the efforts of the library staff and simplify the rental and return process several self-checkout terminals, with embedded RFID readers, were purchased. In this thesis, especially the interactions with these terminals serve as the basis to introduce the concepts and evaluate the applicability of the ITERA test methodology. The first example use case "rental process" was already used throughout the introduction of the methodology. The "return process" is presented in this section and illustrates the usability of the method presented.

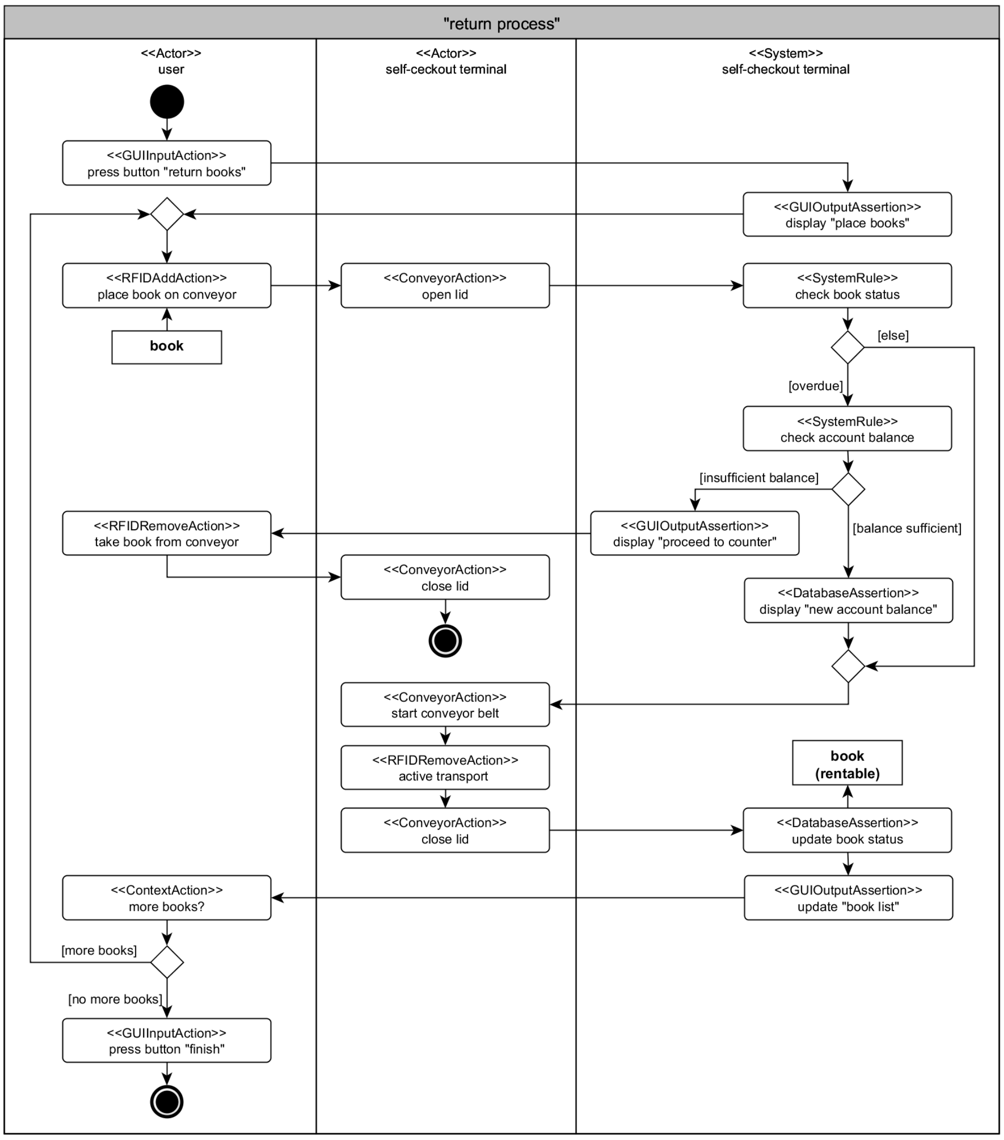

"return process"
<<Actor>> user
<<Actor>> self-ceckout terminal
<<System>> self-checkout terminal
<<GUIInputAction>> press button "return books"
<<GUIOutputAssertion>> display "place books"
<<RFIDAddAction>> place book on conveyor
book
<<ConveyorAction>> open lid
<<SystemRule>> check book status
[else]
[overdue]
<<SystemRule>> check account balance
[insufficient balance]
[balance sufficient]
<<RFIDRemoveAction>> take book from conveyor
<<GUIOutputAssertion>> display "proceed to counter"
<<ConveyorAction>> close lid
<<DatabaseAssertion>> display "new account balance"
<<ConveyorAction>> start conveyor belt
<<RFIDRemoveAction>> active transport
book (rentable)
<<ConveyorAction>> close lid
<<DatabaseAssertion>> update book status
<<ContextAction>> more books?
<<GUIOutputAssertion>> update "book list"
[more books]
[no more books]
<<GUIInputAction>> press button "finish"

Figure 6.7: Process Model "return processes"

Figure 6.7 shows the activity diagram of the return processes which already contains the stereotyped activities defined by ITERA meta model. In addition, the meta model has been extended by `<<ConveyorAction>>` to meet special needs of the SUT. Following the control flow of the activity diagram, two GUI interactions have been applied first. These start the return process and display the corresponding message that the terminal is ready. Depending on the actuator of the activity, `<<GUIInputAction>>` (user) or `<<GUIOutputAction>>` (system) was used. The first interaction with the RFID system of the terminal ("place book on conveyor") was typed with `<<RFIDAddAction>>`. Here an RFID equipped book is placed on the shelf of the terminal. The next activity in the control flow represents special feature that was not present in the previous example "rental process" in Section 4.4.2.

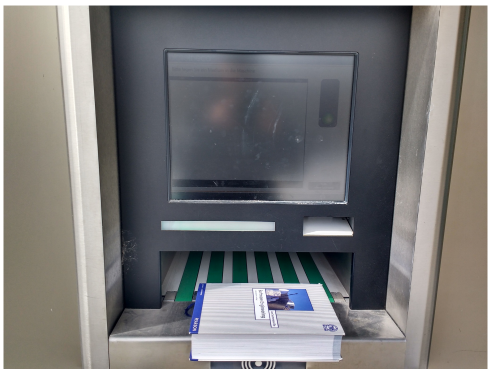


Figure 6.8: 24h-return terminal of the library example

The return terminal (shown in Figure 6.8) is open 24-hours and open to the public. To prevent misuse it can open, respectively close, a return box and uses an conveyor belt. The conveyor belt is used to move the book to the sorting system. In order to model the additional system functionality an second `<<Actor>>` "self-checkout terminal" was used. This is due to the fact that the components of the terminal involve physical activities which consume time. To cover the new components the previously presented ITERA meta model was extended by a stereotype `<<ConveyorAction>>`. This class implements corresponding mock-ups that map the functionality of the actuator within the automatic test environment.

After the RFID reader identified the returned book the lid of the return box is opened and the system performs a check whether the book is overdue or not. Depending on the outcome of this check sometimes a fee has to be paid. This is represented by the activity "check book status". As it contains a decision, the type `<<SystemRule>>` was used here. If a fee has to be paid, additionally the balance of the user's account is checked. Similar to the previous activity the stereotype `<<SystemRule>>` is used and alternative branches of the control flow are modelled. If the balance check was successful, the fee is deduced from the account and the new balance is stored in the database in activity "new account balance" (`<<DatabaseAssertion>>`). If the balance was insufficient the user is asked the proceed to a counter (`<<GUIOutputAssertion>>`) and the book is removed from the shelf "take book from conveyor" (`<<RFIDRemoveAction>>`). The process cannot be completed any more, therefore the process is finished at this point.

Contrary to the other example of the library, used in this thesis, the book is only not removed from the read zone of the RFID reader by the user. If a valid book has been detected, which is not overdue or the balance is sufficient, the system starts the conveyor belt and thus removes the book from the interrogation zone of the RFID reader itself. Therefore, a `RFIDRemoveAction` is located between the "start conveyor belt" and "close lid" activities, both of type `<<ConveyorAction>>`. As a virtual RFID environment is used, this physical movement does not need be directly implemented by the additional actor and it's mock classes. Similar to the other movements of RFID tags, the virtual RFID environment can be instrumented to cover this activity.

Finally, the book is marked as returned in the database. `<<DatabaseAsseertion>>` was used in the activity "update book list", to enable the test execution framework to evaluate the outcome of the test. If more books are to be returned, depicted through the activity "more books?" (`<<ContextAction>>`), the sub-process starting with the acivity "place book on conveyor" is repeated. This is modelled as an cycle in the activity diagram. If not, the process ends by selecting the button "finish" by the user (`<<GUIInputAction>>`).

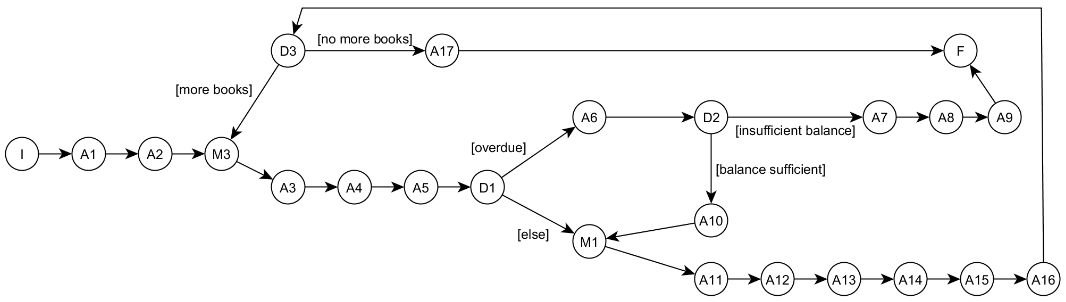


Figure 6.9: ActivityGraph of the return process from the case study library

Figure 6.9 shows the ActivityGraph of the transformed use case "return process" (see Figure 6.7). By traversing the graph with the algorithm introduced in Section 2.2.1 nine test paths are generated. Each of these represent one test case. The textual descriptions of the activities were replaced by A1 to A16. Additionally, the UML guard conditions are shown in the graph on the corresponding edges in square brackets.

TC1: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A7 → A8 → A9 → F
TC2: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → A17 → F
TC3: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A10 → M1 → A11 → A12 → A13 → A14 → A15 → A16
→ D3 → A17 → F
TC4: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A7 → A8 → A9 → F
TC5: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → M3 → A3 → A4 → A5 → D1 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → A17 → F
TC6: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A10 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A7 → A8 → A9 → F
TC7: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A10 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → A17 → F

TC8: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A10 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → M3 → A3 → A4 → A5 → D1 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → A17 → F
TC9: I → A1 → A2 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A10 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → M3 → A3 → A4 → A5 → D1 → A6 → D2 → A10 → M1 → A11 → A12 → A13 → A14 → A15 → A16 → D3 → A17 → F

To improve readability and comprehensibility a compact representation of the generated test cases, only containing the guard conditions, is given in Table 6.2.

TC	**Conditions**
1	[overdue] → [insufficient balance]
2	[else] → [no more books]
3	[overdue] → [balance sufficient] → [no more books]
4	[else] → [more books] → [overdue] → [insufficient balance]
5	[else] → [more books] → [else] → [no more books]
6	[overdue] → [balance sufficient] → [more books] → [overdue] → [insufficient balance]
7	[else] → [more books] → [overdue] → [balance sufficient] → [no more books]
8	[overdue] → [balance sufficient] → [more books] → [else] → [no more books]
9	[overdue] → [balance sufficient] → [more books] → [overdue] → [balance sufficient] → [no more books]

Table 6.2: Shortened test paths generated during traversal of the ActivityGraph of the “return process”

Test case 2 represents the main success scenario of the use case. Here only one book which is not overdue is returned by the user. Accordingly, test case 5 can be treated as an extension of the main success scenario through a cycle. In this test case two books are returned successfully. All of the other test cases represent alternative execution paths with variations of the normal process, where the book to be returned is overdue and either the account balance is sufficient to cover the fee or not.

It is easy to see, that the coverage criteria C_{ACD} has been fulfilled by the generated test cases. As a reminder, the Activity Concurrent Coverage Criteria requires that at least one permissible sequence of concurrent activities is contained and that every cycle is run at least once. Concurrent activities are not part of the original activity diagram (see Figure 6.7) and every cycle contained is traversed at least once (see test case 4-9).

Considering the given conditions for each test case, the tester can now select appropriate test input data. For example, in test case 1 a book that is already over the loan period as well as a user who does not have enough credit on his account to cover the fee in full. Accordingly, a book whose loan period has not yet been exceeded is selected for test case 3.

6.2.3 Summary and Results

Due to the use case scenarios of the example application "retail store", examined in the first feasibility study, the test generated from the model consists of only a few instructions. However, the ability to model different use cases is limited only by the power of the description by UML activity diagrams. These have been applied in a wide range of different software applications, so it is to be expected that much more complex use cases can also be modelled with the methodology presented in this thesis.

The advantage of automatically generating the test cases for the application case recedes in the first case study performed behind the advantages of automating test execution. The test could have been created manually in this form with small effort. However, the aim of the feasibility study was to demonstrate applicability of the methodology and that the technology for test automation with simulation of the RFID environment presented in this thesis is supported by the creation of systematic tests for RFID applications. Especially the test automation technology proposed in this thesis allows for an efficient test process or even makes it possible to test the application examined in the case study adequately and automatically.

During the execution of the feasibility study some weaknesses of the ITERA methodology have been discovered. These can be divided into 3 categories:

- Contradicting conditions
  The conditions at the `DecisionNodes` of the Activity Diagram specified during test case design may be mutually exclusive. Currently, the methodology does not verify the satisfiability of nested conditions. As a result the test case generation algorithm might produce invalid test paths that have to be resolved manually. However, this already hints at possible misconceptions during the specification of the application under test.
- Consistency
  The test data and test cases are strongly depended on each other. If either the test case, respectively the activity diagram, or the test data is altered, the tester needs to manually check if the consistency between both is conserved. However, adding a traceability attribute might enable a fast detection of inconsistencies.
- Undetectable tags and probabilistic reads
  Test including scenarios where RFID tags can not be read, cannot be designed with the proposed methodology. Currently, a tag moves along an activity flow until the end node is reached. Situations where tags might stay undetected at a reader or can only be read with a certain probability can not be constructed. However, this might be solved by extending the methodology to reflect probabilities through a property of the `<<RFIDAddAction>>`.

However, the evaluation of the applicability of the proposed method showed that typical RFID Applications can be modelled with the ITERA method. Although the first two classes of RFID applications generate simplistic test cases, the example library shows that more complex RFID application scenarios exist. However, the investigation of test case generation in the second feasibility study clearly demonstrates that the principles of method presented in this thesis can also be used for much more complex use cases. It is to be expected, that the complexity of RFID use cases will experience strong growth with regard to application scenarios in the context of Industry 4.0.

6.3 Evaluation of the modified Algorithm

Subject of this validation is the modified BFS algorithm for test case generation (see Section 4.5). It provides evidence that the algorithm satisfies the Concurrent Activity Coverage criteria and produces valid test cases.

To perform a validation, UML activity diagram fragments with low complexity were created and successively assembled into complex models. The algorithm for generating test cases was applied to these models and fragments and results were checked for completeness and correctness. The result was also checked with regard to the C_{CAD} coverage criterion (Concurrent Activity Coverage). The aim of this procedure is to provide evidence that the test case generation algorithm generates correct test cases from the UML activity diagrams refined through the ITERA meta-model. Criteria for the evaluations are on the one hand the fulfilment of the required coverage criterion C_{ACD} and on the other hand the completeness and correctness of the resulting test cases by manual inspection. This will be subject in Section 6.3.

The test method for RFID applications investigated in this thesis comprises several steps, which are roughly divided into the phases of test case modelling, test case generation and test case execution. This section is intended to provide evidence that the algorithm for generating abstract test cases (see Section 4.5.3) generates valid test cases, i.e. test cases in which simulated interactions with the SUT only take place in a valid sequence and which also satisfy the concurrent activity coverage criterion C_{CAD} defined in Section 2.2.1.

Even for activity diagrams with relatively low complexity, tests can be created that make it difficult to check the correctness of the results by manual checking. This is due to concurrent and cyclic structures in the control flow and successive mutually exclusive conditions specified during refinement of the test case specification. Nevertheless, it is possible to manually check whether the generated execution paths are valid sequences in the sense of the underlying activity diagram. For this it has to be checked if the contained test steps occur in exactly the order specified in the activity diagram for each test case. For complex models, however, it is no longer possible to check whether all generated execution paths are actually covered by the corresponding sequence in the activity diagram by manual inspection.

One strategy for validating complex models is to identify modelling patterns of low complexity for which manual inspection can be used to confirm their correctness. This assumes that complex models are composed of model fragments of lower complexity, each of which can be assigned to a individual modelling pattern. Inversely, complex models can then be broken down into components of lower complexity. If it can be shown that model fragments (patterns) of low complexity are correctly translated by the algorithm and it can be shown that even the combination of simple patterns to more complex models is correct, it is obvious that the output is valid even for models of any complexity. Thus, the validation of the algorithm and its implementation is based on the demonstration that fragments of UML activity diagrams with low complexity are correctly converted into test cases in accordance with the coverage criterion. In a second step, the basic model fragments are successively combined into more complex models and it is shown that these more complex models are also produce correct test cases.

The aim is therefore to demonstrate that only valid sequences of individual test steps are generated, i.e. that there are no interchanges between two test steps compared to their original sequence in the UML activity diagram. Accordingly, it should be shown that concurrent sequences are mapped to non-concurrent sequences and cycles in the control flow of the UML activity diagram are also traversed in accordance with the coverage criterion defined in Section 2.2.1.

Staines [132] and Wohed ed. al [133, 134] identify possible atomic modelling patterns that can be used to validate the test case generation algorithm. The authors identify the following modelling patterns for UML activity diagrams (1) activity sequence, (2) activity parallelization/synchronisation, (3) alternative branching, (4) merging alternative branches, and (5) iteration. Staines also supports the thesis that complex UML activity diagrams can be created by repeatedly combining (i.e. refining) these patterns.

6.3.1 Activity Sequence

The simplest modelling pattern that can be identified for activity diagrams is a sequence of activities. Embedding a sequence of length one into the empty activity diagram defines the minimum syntactically correct, non-empty activity diagram (Figure 6.10), which specifies only one activity in addition to start (UML InitialNode) and end (UML ActivityFinalNode) nodes. This minimal model is the starting point for combining complex activity diagrams by repeatedly applying replacement operations of the modelling pattern sequence of length one with complex modelling patterns.

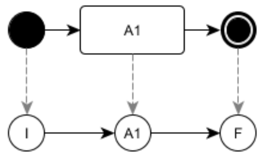


Figure 6.10: Activity diagram and AG of pattern sequence (length one)

Applying the proposed algorithm, of the ITERA method, the following test path is generated:

$$TC1 : I \rightarrow A_1 \rightarrow F$$

The correctness of the minimum UML activity diagram to an execution path is thus validated, since it can be easily seen that the algorithm generates exactly the execution sequence of test steps required by the coverage criterion C_{ACD} for this activity diagram.

Similar to the extraction of the minimum activity diagram from the modelling pattern activity sequence of length one, activity diagrams can also be derived from the modelling pattern activity sequence of any length, which model several activities in a clearly defined execution sequence. Figure 6.11 shows an activity diagram that adds exactly one more activity to the minimum activity diagram.

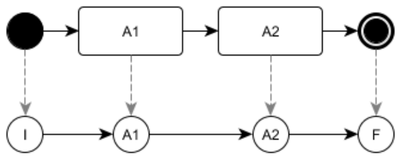


Figure 6.11: Activity diagram and AG of pattern sequence (length two)

Using the introduced algorithm on the ActivityGraph of this activity diagram the following test path is generated:

$$TC1 : I \to A_1 \to A_2 \to F$$

When creating the execution path, this activity diagram also only requires the correct sequence of activities, which confirms the correctness of the result of the test case generation for activity sequences of length two. In the spirit of inductive reasoning, the pattern activity diagram of length one defines the initial premise, the activity diagram of length two is the induction step, whereby the correctness is also justified for activity diagrams of any length.

6.3.2 Activity Branch

Another modelling pattern in UML activity diagrams is the disjunctive control flow branching. Similar to the modelling pattern activity sequence, a minimal activity diagram using the mod-

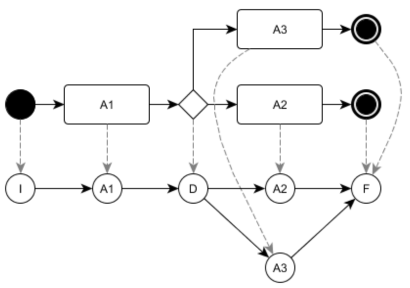


Figure 6.12: Activity diagram and AG of pattern branch

elling pattern activity branch was created for the purpose of validating the model transformation and the algorithm for generating the test cases was applied to it:

$$TC1 : I \rightarrow A_1 \rightarrow D \rightarrow A_2 \rightarrow F$$

$$TC2 : I \rightarrow A_1 \rightarrow D \rightarrow A_3 \rightarrow F$$

Analogous to the modelling pattern activity sequence, the results were checked by manual inspection of the generated test cases. Here, too, the inspection of the paths confirmed the correctness of the test case generation through the proposed algorithm. It was found that the corresponding execution paths were created in accordance to the C_{ACD} coverage criteria for the minimum activity diagram shown in Figure 6.12.

The minimum activity diagram from the activity branch model 6.12 is limited to two alternative control flow paths after the decision node. The meta model of UML allows any number of alternative control flow sequences. A model that maps more than two control flow alternatives at a decision node can be traced back to a sequence of several modelling patterns of the type activity branching.

Similar to the already provided plausibility check of the of the modelling pattern activity sequence (Section 6.3.1), the correctness of the algorithm is indirectly demonstrated, as N-decision nodes can be modelled through the use of n decisions nodes in sequence. However, for completeness the algorithm was also used on a model with three alternative control flows:

$$TC1 : I \rightarrow A_1 \rightarrow D \rightarrow A_2 \rightarrow F$$

$$TC2 : I \rightarrow A_1 \rightarrow D \rightarrow A_3 \rightarrow F$$

$$TC3 : I \rightarrow A_1 \rightarrow D \rightarrow A_4 \rightarrow F$$

6.3.3 Activity Merge

The counterpart to the modelling pattern activity branch, identified by Wohed et al. is a disjunctive merge. It continues different control flows on a common control flow path. The aim of the validation of the test case algorithm is to provide evidence that correct test cases are generated from the disjunctive merge in the sense of the coverage criterion. Figure 6.13 shows

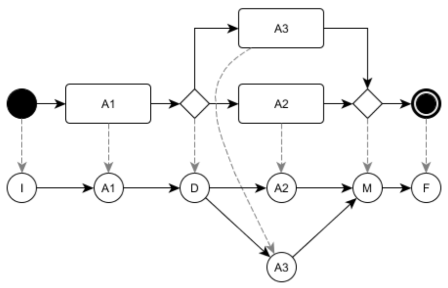


Figure 6.13: Activity diagram and AG of pattern merge

the modelling pattern activity merge. The UML MergeNode is characterized by the fact that, unlike the UML JoinNode modelling element (see Section 6.3.4), the control flow does not block. According to the semantics of UML, a MergeNode does not have to wait until there are tokens on all incoming edges. Similar to the modelling pattern of disjunctive branching, it is possible to unite more than just two control flow sections using a UML MergeNode element.

$$TC1 : I \rightarrow A_1 \rightarrow D \rightarrow A_2 \rightarrow M \rightarrow F$$

$$TC2 : I \rightarrow A_1 \rightarrow D \rightarrow A_3 \rightarrow M \rightarrow F$$

The evaluation of the generated test cases shows, that the correct number of paths in the right sequence of activities have been generated, thus the algorithm for traversing the activity graph containing the modelling element activity branch is implemented correctly.

6.3.4 Activity Concurrency

Another modelling pattern is conjunctive branching, which is modelled as ForkNode in the UML syntax. In the semantics of test execution, the use of this modelling pattern means that activities on the concurrent control flows are mixed but not swapped in sequence. The prerequisite for fulfilling the coverage criterion is the assumption that a user is not able to interact with several components of an application at the same time. During testing, concurrent test steps are therefore interleaved. According to the restriction of the activity diagram in this thesis, this modelling element may only occur in pairs with a conjunctive union. To model this pattern, UML provides the JoinNode element. In the semantics of activity diagrams, this pattern represents a synchronization of concurrent control flows, whereby the continuation of the control flow is blocked until a token is present at all incoming control flows.

In the semantics of testing an application, the use of this modelling pattern in the control flow diagram means that up to a UML JoinNode element, activities modelled in parallel may occur mixed but not reversed in their sequence, and the control flow after the UML ForkNode element is preceded in a single strand without local concurrency. The purpose of the validation of the transformation of UML activity diagrams based on the modelling pattern concurrency is to demonstrate that only those execution sequences of activities are generated that are permitted according to the coverage criterion defined in Section 2.2.1. In particular, it must be shown that no execution sequences are generated, in which activities originally on concurrent control flows before the JoinNode, occur in the execution path generated by the transformation after the JoinNode.

The activity pattern concurrency is shown in Figure 6.14 by the corresponding activity diagram. A special aspect of the chosen validation sample is the odd number of nodes in the concurrent section, due to the algorithm's preference for shorter paths. It must be checked whether the test case generation generates exclusively permissible execution sequences in the sense of the coverage criterion.

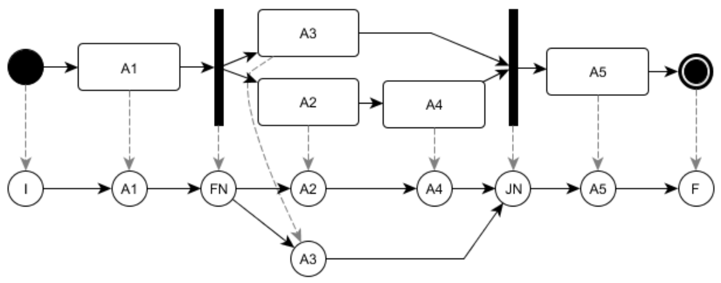


Figure 6.14: Activity diagram and AG of pattern concurrency

The application of the introduced algorithm generated the following test case:

$$TC1 : I \rightarrow A_1 \rightarrow FN \rightarrow A_2 \rightarrow A_3 \rightarrow A_4 \rightarrow JN \rightarrow A_5 \rightarrow F$$

Through manual evaluation respecting the precedence rules (see Section 2.2.1), it is shown that a correct sequence of activities has been derived. Thus, the algorithm for traversing the activity graph containing the modelling element activity concurrency produces correct test cases.

Similar to the previous patterns, an element of type ForkNode can have more than two outgoing control flows. As with the other patterns, this situation can be modelled as disjunctive branching by connecting several ForkNode elements in series, each with two outgoing edges. To ensure that only permissible permutations of concurrent elements are contained in the set of generated execution paths, a precedence test routine was used to ensure that no impermissible partial sequences are contained.

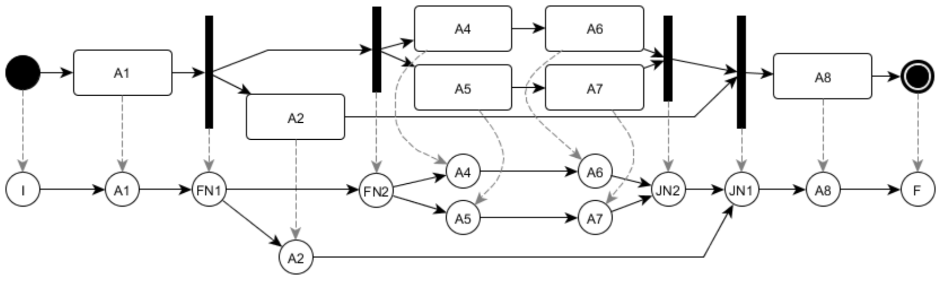


Figure 6.15: Activity diagram and AG of pattern nested concurrency

Applying the proposed algorithm on the nested concurrency pattern (Figure 6.15) the following execution path was generated:

$$TC1 : I \rightarrow A_1 \rightarrow FN_1 \rightarrow FN_2 \rightarrow A_4 \rightarrow A_5 \rightarrow A_6 \rightarrow A_7 \rightarrow JN_2 \rightarrow A_2 \rightarrow JN_1 \rightarrow A_8 \rightarrow F$$

Similar to the previous evaluations it was ensured through manual inspection, that the precedence rules of the output are maintained. In particular, that the generated execution path does not contain any invalid sequences, in which activities of the concurrent control flow occur in reversed order before the join. Furthermore, the verification ensured that only permissible execution sequences were generated from the UML activity diagram shown in Figures 6.14 and 6.15 according to the concurrent activity path coverage criterion C_{CAD} defined in Section 2.2.1.

6.3.5 Validation of composite models

The strategy for validating the transformation algorithm is based on the premise that complex models can always be interpreted as combinations of models of lower complexity. The following sections demonstrate that higher models can be generated by combining the modelling patterns defined in the preceding section. In addition, it is shown that the algorithm presented in Section 4.5.3 also correctly converts complex models into valid test cases. The combination of atomic modelling patterns to complex models is based on substitutions, that replace individual activities within an activity diagram by a sub-model or vice versa a sub-model of an activity diagram by an activity of type UML CallBehaviorAction.

Basic Cycle

The cycle, which has already been identified by Staines, also belongs to the composite modelling patterns. The pattern structures the atomic modelling patterns disjunctive branching (Section 6.3.2) and disjunctive merge (Section 6.3.3) in such a way that a cycle arises in the control flow. The cycle can be realized as a while-do or do-while variant, with the difference of the direction of the control flows at the merge or branch nodes. Except the possibility, that a do-while structure can also be iterated zero times, the cycles structurally correspond to each other. However, the evaluation is a comparatively challenging task as the cycles must be executed multiple times, which means that the complexity of the generated test paths quickly increases to a level that can no longer be mastered by a human, especially for nested cycles.

The coverage criterion defined in section 2.2.1 requires that the algorithm for generating tests from UML activity diagrams generates execution paths, such that each cycle in the control flow is executed at least once. The validation of the algorithm therefore aims to demonstrate that it generates correct results (execution paths) for this requirement. Analogous to the validation of

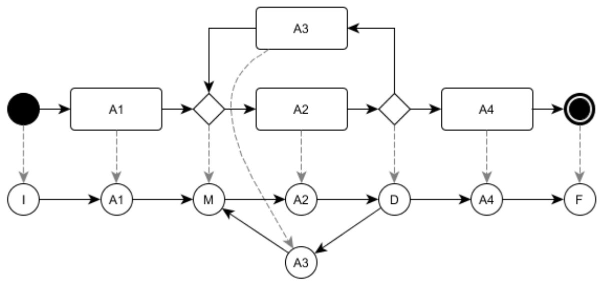


Figure 6.16: Activity diagram and AG of an basic cycle

the atomic modelling patterns, a minimal activity diagram is created for the validation of the modelling pattern cycle (Figure 6.16). Here a variant of the activity diagram was deliberately chosen, in which an activity is in the front (node $A1$) and after the cycle (node $A4$). This activity diagram is used to show the correctness for different variations where the nodes A_2 and A_3 are refined by other modelling patterns, in particular by further cycles.

The activity diagram shown in Figure 6.16 was used to validate the test case generation algorithm. For the coverage criterion concurrent activity path, two execution paths are expected from the activity diagram: one that evaluates the branch at node D so that the cycle is not traversed, and one that evaluates the branch so that the cycle is executed.

$$TC1 : I \rightarrow A1 \rightarrow M \rightarrow A2 \rightarrow D \rightarrow A4 \rightarrow F$$

$$TC2 : I \rightarrow A1 \rightarrow M \rightarrow A2 \rightarrow D \rightarrow A3 \rightarrow M \rightarrow A2 \rightarrow D \rightarrow A4 \rightarrow F$$

An inspection of the activity diagram shown in Figure 6.16 has confirmed that the algorithm generates correct execution paths in accordance to the concurrent activity path coverage criterion. The execution paths illustrated show that the algorithm has actually generated two execution paths for the activity diagram. Test Case $TC2$ represents the cycle initiated at node D, which returns the control flow via node A3 to node M, so that node A2 is executed several times. The result meets the requirement of the coverage criterion, which shows the correctness of the transformation for the base variant modelling pattern cycle.

The example of a cyclic activity diagram shown initially only covers the base case on which only a single activity is modeled. The aim of validation is to provide evidence that models of any complexity based on the pattern cycle are also transformed to test cases in conformity with the coverage criterion. Figure 6.17 shows the modified activity diagram. By refining node A3, the

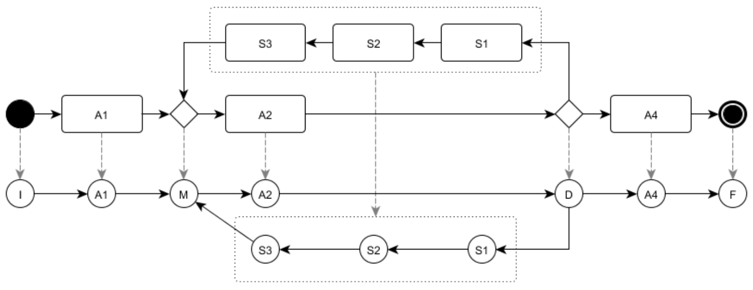


Figure 6.17: Complex activity diagram and AG of a cycle with a sequence

original activity diagram in Figure 6.16 has been extended to a more complex model. The model has been extended by a sequence whose correct test case generation has already been shown.

To still meet the coverage criterion, two execution paths must be created for this model, each representing an execution path with or without cycle execution. Furthermore, the execution paths generated from the model in Figure 6.17 may only differ at the point of the refinement of the model, i.e. specifically only at the path where the added sequence is modelled.

$TC1 : I \rightarrow A1 \rightarrow M \rightarrow A2 \rightarrow D \rightarrow A4 \rightarrow F$
$TC2 : I \rightarrow A1 \rightarrow M \rightarrow A2 \rightarrow D \rightarrow S1 \rightarrow S2 \rightarrow S3 \rightarrow M \rightarrow A2 \rightarrow D \rightarrow A4 \rightarrow F$

Ihe result is compliant with the coverage criterion, the algorithm has generated two execution paths, each representing a run with and without a cycle. Test case 2 shows that the node sequences S1, S2, S3 were also correctly placed and, compared to the simple cycle, the execution paths differ only in the area where the activity diagram was refined by inserting a sequence.

Nested Cyle

It has already been shown that the algorithm for simple cycles produces correct results. The atomic modelling patterns of disjunctive branching and disjunctive union can also be used in such a way that nested cycles are constructed. Figure 6.18 shows the activity diagram already

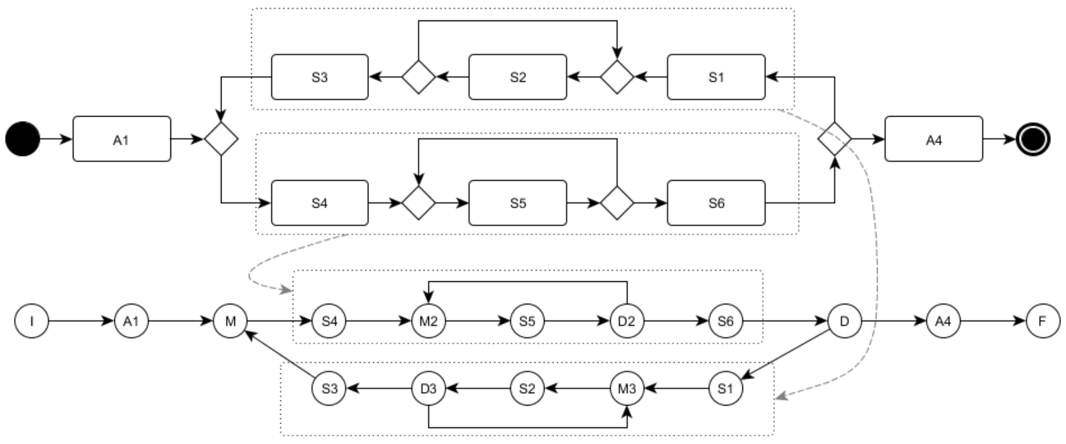


Figure 6.18: Complex activity diagram and AG of nested cycles

shown in Figure 6.16 again, but the nodes A2 and A3 have been refined with another instance of the activity diagram shown in Figure 6.16. Analogous to the procedure for validating the modelling pattern activity concurrency, the approach of refining the model to be validated with itself was also chosen here.

The algorithm for generating test cases was applied to this model and the results were subjected to a manual inspection. The generated execution paths are listed below:

$TC1 : I \rightarrow A1 \rightarrow M \rightarrow S4 \rightarrow M2 \rightarrow S5 \rightarrow D2 \rightarrow S6 \rightarrow D \rightarrow A4 \rightarrow F$
$TC2 : I \rightarrow A1 \rightarrow M \rightarrow S4 \rightarrow M2 \rightarrow S5 \rightarrow D2 \rightarrow M2 \rightarrow S5 \rightarrow D2 \rightarrow S6 \rightarrow D \rightarrow A4 \rightarrow F$
$TC3 : I \rightarrow A1 \rightarrow M \rightarrow S4 \rightarrow M2 \rightarrow S5 \rightarrow D2 \rightarrow S6 \rightarrow D \rightarrow S1 \rightarrow M3 \rightarrow S2 \rightarrow D3 \rightarrow S3 \rightarrow$
$M \rightarrow S4 \rightarrow M2 \rightarrow S5 \rightarrow D2 \rightarrow S6 \rightarrow D \rightarrow A4 \rightarrow F$
$TC4 : I \rightarrow A1 \rightarrow M \rightarrow S4 \rightarrow M2 \rightarrow S5 \rightarrow D2 \rightarrow S6 \rightarrow D \rightarrow S1 \rightarrow M3 \rightarrow S2 \rightarrow D3 \rightarrow M3 \rightarrow$
$S2 \rightarrow D3 \rightarrow S3 \rightarrow M \rightarrow S4 \rightarrow M2 \rightarrow S5 \rightarrow D2 \rightarrow S6 \rightarrow D \rightarrow A4 \rightarrow F$

It can be seen that all cycles, also in combination with each other, are in conformity with the coverage criterion and thus covered by an execution path in the activity diagram, which supports the correctness of the test case generation in terms of the coverage criterion.

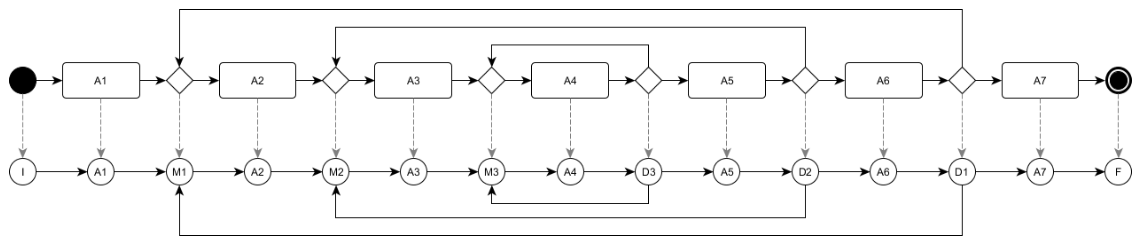


Figure 6.19: Complex activity diagram and AG with multiple nested cycles

To ensure that the test case generation algorithm generates correct results even for multiple nested cycles, another variant of the activity diagram shown in Figure 6.19 was created in a further validation step. Here the algorithm was applied two times with different depths, which allows to specify how often cycles are traversed.

Nested depth 1:

$TC1 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC2 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC3 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC4 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

Nested depth 2:

$TC1 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC2 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC3 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC4 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC5 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC5 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC6 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC7 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

$TC8 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$
$TC9 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$
$TC10 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$
$TC11 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$
$TC12 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow M3 \rightarrow A4 \rightarrow D3 \rightarrow A5 \rightarrow D2 \rightarrow A6 \rightarrow D1 \rightarrow A7 \rightarrow F$

Especially the traversal with increased depth represents a special cognitive performance for a human tester, both in manual test case creation and in its execution. It contains the run-through of all cycles occurring in the model, if necessary in combination with each other. The tester must create the valid preconditions in each step, perform a number of test steps (i.e. enter test data, place the corresponding RFID tags in the reading field) and then check the postconditions.

A comparison of the results with the activity diagrams depicted in Figure 6.19 shows that, the transformation algorithm is stable against nested embedding of cycles. In particular, all execution paths (double and triple nested cycles) that were generated are in accordance with the concurrent activity coverage criteria C_{ACD}.

Overlapping cycles

The atomic modelling patterns of disjunctive branching and disjunctive merge can also be linked together in such a way that overlapping cycles occur, shown in Figure 6.20. Here, a superficially simple model can quickly rise to a complex set of alternative execution paths through the modelled system, each of which must be addressed in an individual test.

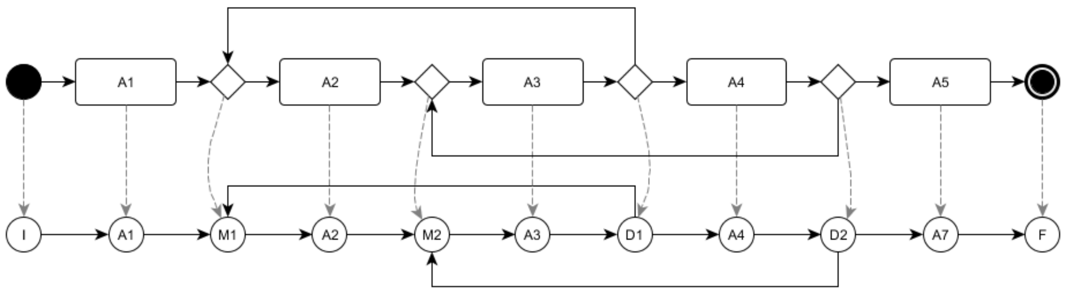


Figure 6.20: Complex activity diagram and AG with overlapping cycles

In a similar procedure to the previous samples, the algorithm for generating test cases produces the following results:

$TC1 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow D1 \rightarrow A4 \rightarrow D2 \rightarrow A7 \rightarrow F$
$TC2 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow D1 \rightarrow A4 \rightarrow D2 \rightarrow A7 \rightarrow F$
$TC3 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow M2 \rightarrow A3 \rightarrow D1 \rightarrow A4 \rightarrow D2 \rightarrow M2 \rightarrow A3 \rightarrow D1 \rightarrow A4 \rightarrow D2 \rightarrow A7 \rightarrow F$

By manual inspection it can be seen that the transformation algorithm generates correct results in terms of the coverage criterion for the model shown in Figure 6.20. This demonstrates that the algorithm is also correct for overlapping cycles.

Interlinked cycles

Another driver for complexity is based on the pattern interlinked or concatenated cycles. Figure 6.21 shows an example of of this pattern. In order to evaluate the algorithm it was also applied to this model.

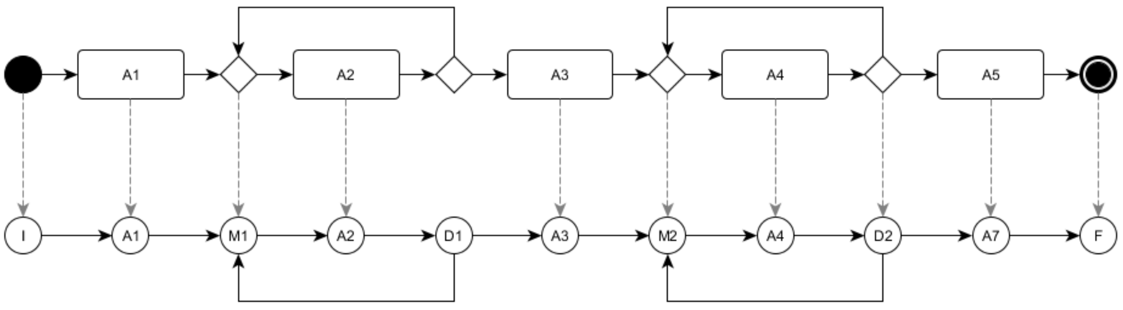


Figure 6.21: Complex activity diagram and AG with interlinked cycles

Figure 6.21 shows a modification of the activity diagram already introduced in Figure 6.16. Specifically, the activity diagram depicted here is also a variant of the modelling pattern sequence (see Section 6.3.1 in Figure 6.11) of length two, were both activities have been refined by a cyclic structure.

The execution sequences generated from the activity diagram shown in Figure 6.21 are shown below. All three execution paths run through the activity diagram from the start node to the end node and all activities are represented by test cases in the modelled order. As required by the coverage criterion, all existing cycles in the model are run at least once and thus fulfil the C_{ACD}.

$TC1 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow D1 \rightarrow A3 \rightarrow M2 \rightarrow A4 \rightarrow D2 \rightarrow A7 \rightarrow F$
$TC2 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow D1 \rightarrow M1 \rightarrow A2 \rightarrow D1 \rightarrow A3 \rightarrow M2 \rightarrow A4 \rightarrow D2 \rightarrow A7 \rightarrow F$

$TC3 : I \rightarrow A1 \rightarrow M1 \rightarrow A2 \rightarrow D1 \rightarrow A3 \rightarrow M2 \rightarrow A4 \rightarrow D2 \rightarrow M2 \rightarrow A4 \rightarrow D2 \rightarrow A7 \rightarrow F$

6.3.6 Summary and Results

In the preceding sections, the identified atomic modelling patterns were combined to more complex modelling patterns. With the UML CallBehaviorAction modelling element, the UML meta model offers the possibility of coarsening partial models of UML activity diagrams or refining individual activities by partial models, whereby each non-atomic activity diagram can be reduced to a single activity on the discussed atomic modelling patterns.

Based on this feature of the UML meta model, the previous sections showed that the algorithm for systematic test case determination developed in this thesis also generates correct results for complex models in terms of the coverage criterion.

A validation for complex models by manual inspection of the generated test paths can usually only barley be performed because the complexity of the test cases quickly becomes unmanageable for humans. Therefore, the generation of test cases was exercised with models that were specifically constructed in such a way that the resulting test cases are simplistic and allow conclusions to be drawn about the behaviour of the test case generation in more complex models.

In all cases, correct execution paths were generated from the activity diagrams according to the coverage criterion C_{CAD}. Due to the traversal of the presented algorithm it is concluded that the test case generation generates correct results even if models are further refined.

6.4 Discussion

The evaluation of the ITERA test method is one of the most challenging tasks of this research. As Dijkstra already stated, testing a program can be a very effective way to detect faults, but it is not suitable for showing the absence of faults. Thus, the ratio of the absolute number of faults to the faults detected by the test within a program can generally hardly be used as a benchmark for evaluating a test method. In addition, due to the enormous variety of RFID applications, it is difficult to evaluate the test method using some example applications, since it is generally not possible to infer from the test results of one application to another. Furthermore, during the study of the method presented here, not enough case studies were available to make a significant statistical statement. A comparison with other RFID test methods is likewise not possible due to the absence of adequate approaches.

However, an evaluation in regard to practical benefits in terms of an subjective evaluation of the impacts was carried out to demonstrate the benefits of using the proposed test methodology:

- RFID specific challenges
  The specific requirements for testing RFID applications (Chapter 3) have been considered during the development of the methodology. To cope with these specific needs an enhancement of the acticity diagram has been proposed in order to include RFID specific details, like timing and test input data, through the ITERA UML profile (meta model, see Section 4.3.1). Additionally, a systematic method to determine test cases, which also considers the unreliable nature of RFID applications in form of robustness tests have been presented (see Section 4.6.3). The costs that arise during test execution, are addressed by the use of a virtual RFID test environment, which not only allows to execute the tests automatically, but, also makes the use of laboratory environments unnecessary. Moreover it provides instantly available RFID readers, whose counterparts might not be deployed in the final RFID system, yet. Furthermore, the method presented in this dissertation enabled the automation of tests, that would otherwise have been impossible to perform in regard to test reproducibility and could only have been carried out in a laboratory environment with great manual effort.

- Variety of RFID applications
  The applicability of the methodology is largely dependent on two factors. (1) Applicability of the activity diagram to describe the use case and the (2) variety of interfaces of the system under test, e.g. additional sensors to complement the RFID system. With regard to the evaluation of the method presented, it is therefore necessary to examine whether these factors have been sufficiently taken into account.

  The variety of different application areas of RFID technology (Section 2.3.2) are taken into account through the build-in possibility to extent the presented meta-model. This allows the application of the methodology in other scenarios as described in the example use cases and supports the integration of additional sensors, for example light barriers or weight scales, as far as other back-ends for evaluation of the SUT. The applicability of the activity diagram has not been addressed specific, however, it is widely used to design a variety of software and moreover also is barely restricted to computer science topics. Therefore, it can be assumed, that it allows to describe a wide range of RFID application scenarios and can thus be used for test modelling.

- Evaluation of test case generation
  The model-based method used to determine the test cases using a control flow is generally widely used. The group of control flow-oriented test methods belongs to the structure-oriented test methods and are not just used in black-box tests, but are also employed in the area of white box tests (based on the source code). Therefore, it can be assumed for an the evaluation of the presented test methodology, that the procedure is basically suitable to systematically create test cases. However, the quality of control flow oriented test procedures depends decisively on the selected test data. Thus, sorely by fulfilling the C_{CAD} overlap criterion, which the test cases created in this work meet, a reliable statement about the correctness of a software cannot be made. However, it is generally

accepted that a structured systematic approach to test creation is better suited than the random selection of test cases.

- Comparison to other approaches
  The approach to testing RFID applications presented in this thesis combines the systematic model-based generation of tests with the automated execution of tests and the simulation of RFID environmental parameters in a common concept. The approach therefore offers an advantage over other technologies, as they do not cover the full range of functions. With regard to the functionality this advantage is objectively measurable: the presented methodology enables an automation of test case creation, its automated execution and the simulation of RFID environments within a consistent and coherent toolbox (i.e. UML profile for modelling context and test data, transformation tool for test case generation, adapted RFID environment model). The approaches examined in the chapters 2.2.2 and 2.3.5 differ either fundamentally with regard to the objective (RFID test data generation, physical effects, etc.) or in their applicability to RFID specific Testing (UML test case generation). A comparable concept does not exist at present. The approaches on test data generation are only based on models describing the flow of RFID tags. The application and especially the state of the RFID application is not considered at all. However, by using the ITERA methodology, not only the flow of RFID tags through a system can be modelled, but also the semantic implications for the software. Therefore, the approach presented here can be treated as an superset of the RFID test data generation approaches.

- Comparison to manual testing
  The approach examined in this thesis aims to increase the quality and reduce the complexity of test activities in the scope of effort and systematic testing. The holistic approach presented here, from modelling the SUT, modelling test data, test case creation to test execution with simulation of context parameters, is mostly opposed by manual methods, if they exist at all. Manual testing of RFID applications is, beside that it is very labor intensive, sometimes not even possible. E.g. regarding the uncertainty of RFID tag captureing or that white spots can not be tested reliable or only with enormous efforts and a suitable laboratory environment.

  The advantages over the prevailing manual test execution are objectively measurable, since the presented ITERA concept for the first time provides a methodology for testing RFID applications, which allows to reproduce context information, in particular RFID events, for test execution, which in reality is difficult or not possible at all. In concrete terms, this has the advantage that the simulation of RFID readers enables reliable reproduction of test cases, which makes it possible to generate meaningful test results, especially for regression tests. On the other hand, there is the advantage that tests can be performed with lower costs, since a automated test execution usually works more economically than manual test execution, does not require recovery phases and can be scaled by use of parallelization.

- Target audience

Furthermore, the methodology investigated for the creation and execution of RFID application tests at system level meets the needs of the target group of this work. The target group is the general audience of developers and testers who are involved in the development of RFID applications as part of their work. The desire for methods and tools to decouple testing activities from human labour manifests itself in the desire to increase efficiency on the one hand, and to ensure the quality of testing activities on the other.

Changing requirements are a common phenomenon in software development. If a change in the requirements for an application occurs, this change can be reflected in an application model with relatively little effort. By using the method and technology for test case generation presented in this thesis, the effort required to adapt test cases as a consequence of changing requirements is limited to the effort required to adapt the model and regenerate the test cases. This contrasts with the great effort of manually adapting all test cases.

- Model review
  During the creation of the model, a implicit review of the software under test is performed. Even though this is only a side-effect of the proposed test methodology, it helps to identify misconceptions of the use cases and resulting test cases. This has been experienced during the development of the library example: A book which is reserved usually cannot be rented, however, it is possible to rent this book if the user is the one with the reservation. In the initial test case description this was not included, but it became present when the activity diagram was created.

7 Contribution and Future Work

RFID is a promising technology with a high potential to improve many of today's business processes. Especially with the upcoming demands of Industry 4.0, RFID allows for the realization of a highly customized and autonomous production. Big companies already benefit using this technology, but many SME still struggle to adopt RFID in their businesses, mainly because of economic considerations. To cope with the challenges of RFID in regard to system level testing, a test methodology for systematic test case generation with high automation potentials was presented in this thesis. By using the proposed test methodology costs during development and testing can be decreased. At the same time confidence in the quality of the RFID application can be increased. Both factors have a significant impact on the use of the technology in particular for SME's. But even further, the integrated virtual RFID environment also enables testing without the need of a specialized laboratory equipment and thus enables SME's with limited resources to carry out RFID specific tests.

This chapter will summarize the original work contributed to the body of knowledge and point out suggestions for improvements and future work.

7.1 Contribution

During the development of the individual solutions, a number of topics from software engineering, to modelling and testing RFID applications, have been investigated in detail. The concrete topics of modelling RFID-specific test cases and the derivation of test data, the generation of test cases and their automated execution within a simulated RFID environment have been examined in depth with the focus on a systematic method of deriving tests and suitable technologies to support test automation for RFID applications.

Testing in general - and especially in the context of RFID systems - is the most important activity of analytical quality assurance in practice. On the other hand, however, testing is usually a pragmatic and less systematic activity compared to other disciplines of software engineering. In order to remedy this discrepancy, a novel test procedure - ITERA - was developed, which allows testing of RFID systems in a systematic and transparent way.

A fundamental concept of ITERA is the common modelling of a set of test cases. Test cases are not considered independently of each other. Instead, they are can be modelled in a more general way using UML activity diagrams, which are perfectly suited to describe the process oriented input flow of RFID applications. In combination with the introduced meta model it makes the test specification simple and understandable, even for untrained domain experts. Moreover, it simplifies the clarity and transparency of the overall test process and test coverage achieved.

A common method to evaluate the overall extent of a test suite is a coverage metric. Such measurements at least ensure that essential parts and their functionality have been included in the test to a certain minimum extent. In this thesis such coverage criteria especially tailored for activity diagrams have been proposed. Furthermore, using the ITERA methodology to generate systematic test cases the fulfilment of a coverage metric is already self-contained. The algorithm developed for traversing the test model, does not only allow for automatic test case generation which fulfil the defined coverage criteria, but also addresses challenging tasks during test design such as concurrent activities and complex structures like nested loops.

In order to meet practical requirements, an executable semantics of the test model was emphasized in the development of ITERA. By combining the semantic modelling language for RFID systems, the ITERA meta model and the activity diagram, not only test cases can be derived automatically, but also the prerequisites for executable tests are met. When supported by appropriate test drivers, fully modelled tests can then be executed automatically, which significantly improve the efficiency of the test execution. Furthermore, the test automation helps to shift the focus of examination and testing RFID applications to the creative task of test case design, which is essential for test quality.

The introduction of the RFID systems is often iterative and thus many repeated tests are necessary. With the support of test automation, using Rifidi, it contributes in considerable decrease of time needed for testing and thus increases the acceptance of the test approach. Furthermore, the use of a virtual RFID environment is often the only way to perform a test of RFID applications on a system level, especially in regard to the unpredictable physical environment and the unstable radio field.

In contrast to related work focusing on performance evaluation of RFID middleware [92, 91, 102, 24] and realistic test data generation for EPC-IS [101, 27] a novel test methodology for functional testing of RFID applications on a system level has been presented. Unlike the approach presented in [26, 30, 31], where tests are derived sorely on the movement routes of RFID tagged goods, the work in this thesis also considers the application's state and the implications of the resulting RFID events of the movement routes. Thus, the ITERA test methodology not only allows for systematically deriving tests including test data based on constraints imposed by the execution path, but also considers oracles in order to evaluate the outcome of the test. The test execution framework, including the virtual RFID environment, enhances similar approaches like [92, 89, 90, 91] by the tight integration of the virtual RFID emulator into a test environment which can be easily extended by other mock-up frameworks to support a wide spectrum of RFID application areas.

7.2 Future Work

As part of the research and development of the ITERA method, using UML activity diagrams to develop test case models, derive systematic test cases and the integration of an virtual RFID environment for automated test execution, a number of research questions have arisen that can improve or complement the ITERA methodology.

7.2.1 Enhanced Tool Support

The prototype implementation of the test case generation algorithm developed in this thesis is implemented in Python. Currently, UML activity diagrams which are exported in *Graph Modelling Language* (GML) [135] format are used as valid input for the generation of the test executions paths. Nonetheless, the diagrams can still be created in any UML editor that is capable of exporting the diagram in this format. However, GML is very restricted in comparison the the widely used XMI export format, which also offers a better integration, in particular for the associated class model of the meta model. As part of future work, this could be realized with XMI and a corresponding implementation of the test case generation algorithm.

Additionally, a sophisticated integration of the test case generation algorithm and the modelling of the activity diagram including the UML meta model into a *Integrated Development Environment* (IDE) would be beneficial to ease the development and testing.

Furthermore, the integration of the ITERA methodology into CI environments such as Jenkins[1] can help to supplement the automation process of software provisioning from the IDE to the finished deployment routine including model-based test automation and simulation of the RFID environment. Manual activities could, thus, be reduced to the creation of a model of the SUT and the determination of test data. Manual test procedures, which are necessary to evaluate the specification conformity of different releases of an RFID application, could then be omitted.

7.2.2 Extension of the ITERA Method

The method for model-based test case determination presented in Chapter 4 aims at generating test cases from UML activity diagrams. Determining the appropriate test data to adequately address the system-specific aspects of an SUT, however, is still a semi-automatic activity. During the discussion on test data determination in Section 4.6, the option to generate test data from the predicates (using relational algebra) was already partially addressed. Of particular interest here, could be boundary tests in which the RFID application is specifically stimulated with test data, whose values are at the input range limits. Or even further, a sophisticated method to systematically cover condition coverage in terms of input data could improve the quality of the tests significantly. An efficient strategy to identify or generate such test data is a research area

[1]Jenkins (https://jenkins.io) is an open source automation server, for continuous integration and continuous delivery.

that can further promote the automation of test activities and thus contribute to a optimization of the use of resources in the field of quality assurance.

While the generation of test cases guarantees that all relevant activities and control flows are addressed in tests, there is no guarantee that adequate test coverage is also achieved with regard to the test data. Future research should look how test case modelling must be designed to map different test data for test execution. An open research question here is: “How to efficiently select test data to improve condition coverage for RFID application testing?”. For example, it would make sense for an activity that is executed more than once due to a cycle to be automatically parametrized with different test data for each run. However, this problem cannot be solved trivially, especially for nested cycles and conflicting alternative control flows, because it may be relevant for a concrete test to consider the current state of the application and how often a cycle is executed.

Another field of research that was not considered in this thesis, but is relevant in the field of RFID applications, is the generation of even more realistic environmental conditions of the RFID system. The deviation of simulated physical conditions of the RFID tags captured by the virtual reader would offer the possibility to test other innovative application areas of RFID technology. RFID tag aggregation mechanisms that use the *Returned Signal Strength Indicator* (RSSI) to evaluate the validity of the RFID event or applications using real-time localization could be considered. Probabilistic test methods could prove to be a suitable extension to integrate these into the ITERA methodology. For example test data could be selected according to the predicates of the test case and extended by a random distribution taking into account the unreliability of the radio field.

7.2.3 Extensions with Formal Methods

The formal semantics of RFID systems presented in this thesis are based on time dependent sets. In order to be used as a basis for formal methods, it is necessary to extend the model with a solid logic. This can provide a more accurate representation of operations such as the placement of RFID tags in a RFID readers interrogation zone and the implications for the application's current state. The extension would allow for reasoning of the model itself and could provide a basis to derive well founded assumptions about the consistency of the specification and serve as a foundation for proving correctness.

An interesting approach for further research is therefore the extension of the semantic model of RFID systems presented in Section 4. This raises the question of whether the set-based description can be extended by a formal specification, such as Event B (or the UML-B spin-off) or the Z notation. The use of temporal logic, such as interval temporal logic (ITL), could also be investigated. Starting points are the temporal sets that would receive a formal basis around predicate logical expressions or temporal aspects. If such an extension can be realized, the verification and validation of the model itself or runtime verification could be possible.

References

[1] ISO/IEC, "ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," tech. rep., ISO/IEC, 2010.

[2] K. Finkenzeller, *RFID handbook: Fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication.* Oxford: Wiley-Blackwell, 3 ed., 2010.

[3] Transcends LLC (former Pramari LLC), "Rifidi Emulator Documentation: Developers Guide." online: `http://wiki.rifidi.org/index.php/Engine_Overview`, 2008. [accessed: December 14, 2011].

[4] Bundesministerium für Bildung und Forschung (BMBF), "Industrie 4.0," tech. rep., Bundesministerium für Bildung und Forschung (BMBF), 2017.

[5] A. Dutta, H. L. Lee, and S. Whang, "Rfid and operations management: Technology, value, and incentives," *Production and Operations Management*, vol. 16, pp. 646–655, Sep 2007. Copyright - Copyright Production and Operations Management Society Sep/Oct 2007; Dokumentbestandteil - Diagrams; Graphs; Tables; ; Zuletzt aktualisiert - 2012-02-22; CODEN - POMAEN.

[6] C. Loebbecke, "Rfid technology and applications in the retail supply chain: The early metro group pilot," *BLED 2005 Proceedings*, p. 42, 2005.

[7] C. Loebbecke and J. W. Palmer, "Rfid in the fashion industry: Kaufhof department stores ag and gerry weber international ag, fashion manufacturer.," *MIS Quarterly Executive*, vol. 5, no. 2, 2006.

[8] B. Vijayaraman and B. A. Osyk, "An empirical study of rfid implementation in the warehousing industry," *The International Journal of Logistics Management*, vol. 17, no. 1, pp. 6–20, 2006.

[9] A. J. Fruth, *M.* PhD thesis, Lehrstuhl für Fördertechnik Materialfluss Logistik der Technischen Universität München, 2011.

[10] G. Myers, *The Art of Software Testing, Second Edition.* John Wiley & Sons, Inc., 2004.

[11] M. Buenen and G. Muthukrishnan, "World quality report 2016-2017," *Capgemini, Sogeti und HP*, 2016.

[12] S. Sarma, “Integrating rfid,” *Queue*, vol. 2, pp. 50–57, Oct. 2004.

[13] EPCglobal Inc., *EPCglobal Architecture Framework, v. 1.5*, 2013. online: `http://www.gs1.org/gsmp/kc/epcglobal/architecture`.

[14] C. Bornhövd, T. Lin, S. Haller, and J. Schaper, “Integrating automatic data acquisition with business processes experiences with sap’s auto-id infrastructure,” in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB ’04, pp. 1182–1188, VLDB Endowment, 2004.

[15] A. Huebner, C. Facchi, M. Meyer, and H. Janicke, “Rfid systems from a cyber-physical systems perspective,” in *WISES 2013, the Workshop on Intelligent Systems and Embedded Systems 2013*, 2013.

[16] E. A. Lee, “Cyber physical systems: Design challenges,” in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pp. 363 –369, may 2008.

[17] E. A. Lee, “Cps foundations,” in *Proceedings of the 47th Design Automation Conference*, pp. 737–742, ACM, 2010.

[18] M. Attaran, “Rfid: an enabler of supply chain operations,” *Supply Chain Management: An International Journal*, vol. 12, no. 4, pp. 249–257, 2007.

[19] C. Floerkemeier and E. Fleisch, “Rfid applications: Interfacing with readers,” *Software, IEEE*, vol. 25, pp. 67–70, May 2008.

[20] H. Chen, P. Chou, S. Duri, J. Elliott, J. Reason, and D. Wong, “A model-driven approach to rfid application programming and infrastructure management,” in *e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on*, pp. 256–259, 2005.

[21] V. Derbek, C. Steger, R. Weiss, J. Preishuber-Pflügl, and M. Pistauer, “A uhf rfid measurement and evaluation test system,” *e & i Elektrotechnik und Informationstechnik*, vol. 124, no. 11, pp. 384–390, 2007.

[22] C. Jacinto, M. Romano, A. Montes, P. Sousa, and M. Nunes, “Rfid tuning methodology applied on airport baggage tracking,” in *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pp. 1–4, 2009.

[23] C.-M. Li, “An integrated software platform for rfid-enabled application development,” in *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, vol. 1, pp. 4 pp.–, 2006.

[24] J. Park, W. Ryu, B. Hong, and B. Kim, “Design of toolkit of multiple virtual readers for scalability verification of rfid middleware,” in *The Second International Conference on Emerging Databases (EDB 2010)*, pp. 56–59, 2010.

[25] Q. Jin, Z. Ren, and D. Wang, "Simulating supply chain activities and providing rfid event data for test," in *Information Management, Innovation Management and Industrial Engineering (ICIII), 2011 International Conference on*, vol. 3, pp. 438–441, 2011.

[26] W. Ryu, J. Kwon, and B. Hong, "An rsn tool: A test dataset generator for evaluating rfid middleware," in *Secure and Trust Computing, Data Management and Applications* (J. Park, J. Lopez, S.-S. Yeo, T. Shon, and D. Taniar, eds.), vol. 186 of *Communications in Computer and Information Science*, pp. 217–224, Springer Berlin Heidelberg, 2011.

[27] J. Mueller, C. Popke, M. Urbat, and H. Zeier, A. andPlattner, "A simulation of the pharmaceutical supply chain to provide realistic test data," in *Advances in System Simulation, 2009. SIMUL '09. First International Conference on*, pp. 44–49, 2009.

[28] Object Management Group, *UML Specification, v2.2*, 2007. online: `http://www.uml.org/` [accessed: 09,2013].

[29] A. Huebner, C. Facchi, and H. Janicke, "Rifidi toolkit: Virtuality for testing rfid," in *ICSNC 2012 The Seventh International Conference on Systems and Networks Communications*, pp. 1–6, 2012.

[30] W. Ryu, J. Kwon, and B. Hong, "A simulation network model to evaluate rfid middlewares," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 06, pp. 779–801, 2011.

[31] W. Ryu, J. Kwon, and B. Hong, "Generation of rfid test datasets using rsn tool," *Personal and Ubiquitous Computing*, pp. 1–11, 2012.

[32] R. Patton, *Software testing.* Pearson Education India, 2006.

[33] A. Hartman, M. Katara, and S. Olvovsky, "Choosing a test modeling language: A survey," in *Haifa Verification Conference*, pp. 204–218, Springer, 2006.

[34] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[35] A. Abran, J. W. Moore, P. Bourque, and R. Dupuis, eds., *Guide to the Software Engineering Body of Knowledge.* IEEE Computer Society, 2004.

[36] IEE, "IEEE Standard Glossary of Software Engineering Terminology," tech. rep., Institute of Electrical and Electronics Engineers, 1990.

[37] E. W. Dijkstra, "The humble programmer," *Communications of the ACM*, vol. 15, no. 10, pp. 859–866, 1972.

[38] M. R. Lyu, ed., *Handbook of Software Reliability Engineering.* Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[39] A. Spillner, T. Linz, and H. Schaefer, *Software testing foundations: a study guide for the certified tester exam.* Rocky Nook, Inc., 2014.

[40] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[41] B. Beizer, *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., 1995.

[42] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[43] R. Ramler and K. Wolfmaier, "Economic perspectives in test automation: balancing automated and manual testing with opportunity cost," in *Proceedings of the 2006 international workshop on Automation of software test*, pp. 85–91, ACM, 2006.

[44] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*, pp. 85–103, IEEE Computer Society, 2007.

[45] J. Bach, "Test automation snake oil," in *Proceedings of the 14th International Conference and Exposition on Testing Computer Software (TCS'99)*, 1999.

[46] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: empirical observations," *Advances in Software Engineering*, vol. 2010, 2010.

[47] Object Management Group, "Unified modeling language," tech. rep., Object Management Group, 2017.

[48] P. Baker, Z. R. Dai, J. Grabowski, I. Schieferdecker, and C. Williams, *Model-driven testing: Using the UML testing profile*. Springer Science & Business Media, 2007.

[49] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: a systematic review," in *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, pp. 31–36, ACM, 2007.

[50] M. Shirole and R. Kumar, "Uml behavioral model based test case generation: a survey," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–13, 2013.

[51] M. Utting and B. Legeard, *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2010.

[52] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang, "Generating test cases from uml activity diagram based on gray-box method," in *Software Engineering Conference, 2004. 11th Asia-Pacific*, pp. 284–291, IEEE, 2004.

[53] X. Bai, C. P. Lam, and H. Li, "An approach to generate the thin-threads from the uml diagrams," in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pp. 546–552, IEEE, 2004.

[54] H. Kim, S. Kang, J. Baik, and I. Ko, "Test cases generation from uml activity diagrams," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, vol. 3, pp. 556–561, IEEE, 2007.

[55] M. Chen, X. Qiu, W. Xu, L. Wang, J. Zhao, and X. Li, "Uml activity diagram-based automatic test case generation for java programs," *The Computer Journal*, vol. 52, no. 5, pp. 545–556, 2009.

[56] R. D. Ferreira, J. P. Faria, and A. C. Paiva, "Test coverage analysis of uml activity diagrams for interactive systems," in *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pp. 268–273, IEEE, 2010.

[57] A. Holzer, V. Januzaj, S. Kugele, B. Langer, C. Schallhart, M. Tautschnig, and H. Veith, "Seamless testing for models and code," in *International Conference on Fundamental Approaches to Software Engineering*, pp. 278–293, Springer, 2011.

[58] A. Nayak and D. Samanta, "Synthesis of test scenarios using uml activity diagrams," *Software & Systems Modeling*, vol. 10, no. 1, pp. 63–89, 2011.

[59] W. Zhao, Y. Zeng, and L.-w. ZHANG, "Generating test scenarios of functional test from uml activity diagrams," *Computer Engineering and Design*, vol. 22, p. 048, 2006.

[60] A. Heinecke, T. Brückmann, T. Griebe, and V. Gruhn, "Generating test plans for acceptance tests from uml activity diagrams," in *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*, pp. 57–66, IEEE, 2010.

[61] H. T. Jung and S. H. Joo, "Transformation of an activity model into a colored petri net model," in *Trendz in Information Sciences & Computing (TISC), 2010*, pp. 32–37, IEEE, 2010.

[62] R. Seiger and T. Schlegel, "Test modeling for context-aware ubiquitous applications with feature petri nets," in *Proceedings of the Workshop on Model-based Interactive Ubiquitous Systems (MODIQUITOUS)*, 2012.

[63] W. Thanakorncharuwit, S. Kamonsantiroj, and L. Pipanmaekaporn, "Generating test cases from uml activity diagram based on business flow constraints," in *Proceedings of the Fifth International Conference on Network, Communication and Computing*, pp. 155–160, ACM, 2016.

[64] M. Chen, P. Mishra, and D. Kalita, "Coverage-driven automatic test generation for uml activity diagrams," in *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pp. 139–142, ACM, 2008.

[65] M. Chen, P. Mishra, and D. Kalita, “Efficient test case generation for validation of uml activity diagrams,” *Design Automation for embedded systems*, vol. 14, no. 2, pp. 105–130, 2010.

[66] D. Kundu and D. Samanta, “A novel approach to generate test cases from uml activity diagrams.,” *Journal of Object Technology*, vol. 8, no. 3, pp. 65–83, 2009.

[67] H. Li and C. P. Lam, “Using anti-ant-like agents to generate test threads from the uml diagrams,” in *IFIP International Conference on Testing of Communicating Systems*, pp. 69–80, Springer, 2005.

[68] D. Xu, H. Li, and C. P. Lam, “Using adaptive agents to automatically generate test scenarios from the uml activity diagrams,” in *Software Engineering Conference, 2005. APSEC’05. 12th Asia-Pacific*, pp. 8–pp, IEEE, 2005.

[69] A. K. Jena, S. K. Swain, and D. P. Mohapatra, “A novel approach for test case generation from uml activity diagram,” in *Issues and challenges in intelligent computing techniques (ICICT), 2014 international conference on*, pp. 621–629, IEEE, 2014.

[70] C. Mingsong, Q. Xiaokang, and L. Xuandong, “Automatic test case generation for uml activity diagrams,” in *Proceedings of the 2006 international workshop on Automation of software test*, pp. 2–8, ACM, 2006.

[71] L. Briand and Y. Labiche, “A uml-based approach to system testing,” *Software and Systems Modeling*, vol. 1, no. 1, pp. 10–42, 2002.

[72] A. Cavarra, C. Crichton, J. Davies, A. Hartman, and L. Mounier, “Using uml for automatic test generation,” in *Proceedings of ISSTA*, vol. 15, 2002.

[73] H. Robinson, “Obstacles and opportunities for model-based testing in an industrial software environment,” in *Proceedings of the 1st European Conference on Model-Driven Software Engineering, Nuremberg, Germany*, pp. 118–127, 2003.

[74] J. Landt, “Shrouds of time: the history of rfid,” 2001.

[75] J. Landt, “The history of rfid,” *Potentials, IEEE*, vol. 24, no. 4, pp. 8–11, 2005.

[76] C. Floerkemeier and S. Sarma, “An overview of rfid system interfaces and reader protocols,” in *RFID, 2008 IEEE International Conference on*, pp. 232–240, 2008.

[77] S. Leaver, “Evaluating rfid middleware,” tech. rep., Forrester Research, Inc, 2004.

[78] EPCglobal Inc., *EPC Tag Data Standard, v. 1.7*, 2013. online: `http://www.gs1.org/gsmp/kc/epcglobal/tds/`.

[79] EPCglobal Inc., *EPC Tag Data Translation, v. 1.6*, 2011. online: `http://www.gs1.org/gsmp/kc/epcglobal/tdt/`.

[80] EPCglobal Inc., *EPC Class-1 HF RFID Air Interface Protocol, v. 2.0.3*, 2011. online: `http://www.gs1.org/gsmp/kc/epcglobal/hf`.

[81] EPCglobal Inc., *EPC Class-1 Generation-2 UHF RFID Air Interface Protocol, v. 1.2.0*, 2008. online: `http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2/`.

[82] EPCglobal Inc., *EPC Low Level Reader Protocol, v. 1.1*, 2010. online: `http://www.gs1.org/gsmp/kc/epcglobal/llrp`.

[83] EPCglobal Inc., *EPC Discovery, Configuration & Initialisation Standard, v. 1.0*, 2009. online: `http://www.gs1.org/gsmp/kc/epcglobal/dci`.

[84] EPCglobal Inc., *EPC Reader Management Standard, v. 1.0.1*, 2007. online: `http://www.gs1.org/gsmp/kc/epcglobal/rm`.

[85] EPCglobal Inc., *EPC Application Level Events Standard, v. 1.1.1*, 2009. online: `http://www.gs1.org/gsmp/kc/epcglobal/ale`.

[86] EPCglobal Inc., *EPCIS - EPC Information Services Standard, v. 1.0.1*, 2007. online: `http://www.gs1.org/gsmp/kc/epcglobal/epcis`.

[87] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, 2002.

[88] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, "When, how, and why developers (do not) test in their ides," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 179–190, ACM, 2015.

[89] C. Park, W. Ryu, and B. Hong, "Rfid middleware evaluation toolkit based on a virtual reader emulator," in *Emerging Databases, The 1th International Conference on Emerging Databases 2009*, pp. 154–157, 2009.

[90] G. Lee, H. Zhang, C. Park, W. Ryu, and B. Hong, "Design and implementation of virtual test toolkit for testing rfid middleware," in *Intelligent Manufacturing and Logistics Systems, The 6th International Conference on IML 2010*, pp. 1–6, 2010.

[91] H. Zhang, W. Ryu, B. Hong, and C. Park, "A test data generation tool for testing rfid middleware," in *Computers and Industrial Engineering (CIE), 2010 40th International Conference on*, pp. 1–6, July 2010.

[92] L. Jongyoung and K. Naesoo, "Performance test tool for rfid middleware: Parameters, design, implementation, and features," in *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, vol. 1, pp. 149–152, Feb. 2006.

[93] M. D. M. C. E. Palazzi, A. Ceriali, "RFID Emulation in Rifidi Environment," in *International Symposium on Ubiquitous Computing (UCS'09)*, 2009.

[94] J. Siror, S. Huanye, and W. Dong, "Automating customs verification process using rfid technology," in *Digital Content, Multimedia Technology and its Applications (IDC), 2010 6th International Conference on*, pp. 404–409, Aug. 2010.

[95] J. Mueller, M. Schapranow, C. Poepke, M. Urbat, A. Zeier, and H. Plattner, "Best practices for rigorous evaluation of rfid software components," in *RFID Systech 2010, RFID Systech 2010 - European Workshop on Smart Objects: Systems, Technologies and Applications*, 2010.

[96] V. Derbek, A. Janek, C. Steger, J. Preishuber-Pfluegl, and M. Pistauer, "Behavioral model for system level design automation: Passive uhf transponder case study," *IEEE ICST2005, Proceedings*, 2005.

[97] V. Derbek, J. Preishuber-Pflueg, C. Steger, and M. Pistauer, "Architecture for model-based uhf rfid system design verification," in *Circuit Theory and Design, 2005. Proceedings of the 2005 European Conference on*, vol. 2, pp. II–181, IEEE, 2005.

[98] V. Derbek, C. Steger, R. Weiß, D. Wischounig, J. Preishuber-Pfluegl, and M. Pistauer, "Simulation platform for uhf rfid," in *Proceedings of the conference on Design, automation and test in Europe*, pp. 918–923, EDA Consortium, 2007.

[99] C. Semiconductors, "Rfid tag emulator." online: `https://www.cisc.at/?id=25`, 2012. [accessed: June 05, 2012].

[100] A. TaiHyun, K. Gihong, B. Hong, and H. Kwon, "A performance evaluation tool for epcis," in *SENSORCOMM 2011, The 5th International Conference on Sensor Technologies and Applications*, pp. 145–150, 2011.

[101] W. Rui, S. Prives, R. Fischer, M. Salfer, and W. Gunthner, "Data analysis and simulation of auto-id enabled food supply chains based on epcis standard," in *Automation and Logistics (ICAL), 2011 IEEE International Conference on*, pp. 58–63, 2011.

[102] H. Zhang, J. Kwon, and B. Hong, "A graph model based simulation tool for generating rfid streaming data," in *Proceedings of the 13th Asia-Pacific web conference on Web technologies and applications*, APWeb'11, pp. 290–300, Springer-Verlag, 2011.

[103] E. A. Lee, "Computing foundations and practice for cyber-physical systems: A preliminary report," *University of California, Berkeley, Tech. Rep. UCB/EECS-2007-72*, 2007.

[104] E. A. Lee, B. David, T. Martin, and S. S. Sunder, "Cyber-physical systems." online: `http://cyberphysicalsystems.org/`, 2013. [accessed: February 18, 2013].

[105] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference*, DAC '10, (New York, NY, USA), pp. 731–736, ACM, 2010.

[106] Y. Tan, S. Goddard, and L. C. Perez, "A prototype architecture for cyber-physical systems," *ACM SIGBED Review*, vol. 5, no. 1, pp. 1–2, 2008.

[107] C. Talcott, "Cyber-physical systems and events," in *Software-Intensive Systems and New Computing Paradigms*, pp. 101–115, Springer, 2008.

[108] K.-J. Park, R. Zheng, and X. Liu, “Cyber-physical systems: Milestones and research challenges,” *Computer Communications*, vol. 36, no. 1, pp. 1 – 7, 2012.

[109] J. Singh, E. Olsen, K. Vorst, and K. Tripp, “Rfid tag readability issues with palletized loads of consumer goods,” *Packaging Technology and Science*, vol. 22, no. 8, pp. 431–441, 2009.

[110] J. A. Singh, R. Holtz, S. Singh, and K. Saha, “Effect of unitization and product types on readability of tagged packages of consumer goods,” *Journal of Applied Packaging Research*, vol. 2, no. 3, p. 179, 2008.

[111] R. H. Clarke, D. Twede, J. R. Tazelaar, and K. K. Boyer, “Radio frequency identification (rfid) performance: the effect of tag orientation and package contents,” *Packaging Technology and Science*, vol. 19, no. 1, pp. 45–54, 2006.

[112] K. Penttilä, M. Keskilammi, L. Sydänheimo, and M. Kivikoski, “Radio frequency technology for automated manufacturing and logistics control. part 2: Rfid antenna utilisation in industrial applications,” *The International Journal of Advanced Manufacturing Technology*, vol. 31, pp. 116–124, Nov 2006.

[113] Y. Bai, F. Wang, and P. Liu, “Efficiently filtering rfid data streams.,” in *CleanDB*, Citeseer, 2006.

[114] T. Keller, *Mining the Internet of Things: Detection of False-Positive RFID Tag Reads using Low-Level Reader Data*. PhD thesis, University of St. Gallen, School of Management, Economics, Law, Social Sciences and International Affairs, 2011.

[115] EPCglobal Inc., *Core Business Vocabulary, 1.2.2*, 2017. online: `https://www.gs1.org/sites/default/files/docs/epc/CBV-Standard-1-2-2-r-2017-10-12.pdf`.

[116] H. Vogt, “Efficient object identification with passive rfid tags,” in *International Conference on Pervasive Computing*, pp. 98–113, Springer, 2002.

[117] J. Brusey, C. Floerkemeier, M. Harrison, and M. Fletcher, “Reasoning about uncertainty in location identification with rfid,” in *Workshop on Reasoning with Uncertainty in Robotics at IJCAI*, pp. 23–30, 2003.

[118] Y.-J. Tu and S. Piramuthu, “A decision-support model for filtering rfid read data in supply chains,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 2, pp. 268–273, 2011.

[119] Y. Ju Tu and S. Piramuthu, “Reducing false reads in rfid-embedded supply chains,” *Journal of theoretical and applied electronic commerce research*, vol. 3, no. 2, pp. 60–70, 2008.

[120] A. Huebner, C. Facchi, M. Meyer, and H. Janicke, “A model based approach for rfid application testing,” in *IUCC 2013, the IEEE International Conference on Ubiquious Computing and Communications 2013*, 2013.

[121] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[122] E. Codd, "Relational completness of data base sublanguages," *Computer*, 1972.

[123] I. Jacobson, "Object-oriented development in an industrial environment," in *ACM SIGPLAN Notices*, vol. 22, pp. 183–191, ACM, 1987.

[124] A. Cockburn, "Writing effective use cases," *Michigan: Addison-Wesley*, 2001.

[125] H. B. Maynard, G. J. Stegemerten, and J. L. Schwab, "Methods-time measurement.," 1948.

[126] A. Valmari, "The state explosion problem," in *Advanced Course on Petri Nets*, pp. 429–528, Springer, 1996.

[127] D. Peled, "All from one, one for all: on model checking using representatives," in *International Conference on Computer Aided Verification*, pp. 409–423, Springer, 1993.

[128] Transcends LLC (former Pramari LLC), "Rifidi project." online: `http://www.rifidi.org`, 2006. [accessed: December 14, 2011].

[129] Transcends LLC (former Pramari LLC), "Rifidi - from rfidea to business reality." online: `http://sourceforge.net/projects/rifidi/`, 2011. [accessed: March 28, 2011].

[130] Transcends LLC, "Rifidi Statistics." online: `http://sourceforge.net/projects/rifidi/files/stats/timeline?dates=2005-01-05+to+2012-07-02`, 2012. [accessed: June 01, 2012].

[131] G. McCluskey, "Using java reflection." online: `http://java.sun.com/developer/technicalArticles/ALT/Reflection/`, 1998. [accessed: June 12, 2012].

[132] T. S. Staines, "Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets," in *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, pp. 191–200, IEEE, 2008.

[133] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. ter Hofstede, and N. Russell, "Pattern-based analysis of uml activity diagrams," *Beta, Research School for Operations Management and Logistics, Eindhoven*, 2004.

[134] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. ter Hofstede, and N. Russell, "Pattern-based analysis of the control-flow perspective of uml activity diagrams," in *International Conference on Conceptual Modeling*, pp. 63–78, Springer, 2005.

[135] M. Himsolt, "Graphed: A graphical platform for the implementation of graph algorithms (extended abstract and demo)," in *International Symposium on Graph Drawing*, pp. 182–193, Springer, 1994.